Das Lazarus Beispielbuch

Andreas Frieß
Johannes Müller
Die Community von www.lazarusforum.de

6. November 2007

Version SVN $LastChangedRevision: 52 $

Inhaltsverzeichnis

1 Einführung

1.1 Allgemeine Infos

1.2 Entwicklungsumgebung

1.2.1 Tastenkombinationen

Die aktuelle Quelle für die Tastenkombinationen [1] ist die Lazarus-Homepage[10].

Datei	
Tastenkürzel	Erklärung
Strg+O	Datei öffnen
Strg+S	Datei speichern
Umsch+Strg+S	alle Dateien speichern
Strg+P	Drucken

Tabelle 1.1: IDE - Tastenkombinationen Datei

Bearbeiten	
Tastenkürzel	Erklärung
Strg+Z	Rückgängig
Umsch+Strg+Z	Wiederholen
Strg+X	Auswahl ausschneiden
Strg+C	Auswahl kopieren
Strg+V	Auswahl einfügen
Strg+I	Auswahl einrücken
Strg+U	Auswahl ausrücken
Strg+A	Alles auswählen
Umsch+Strg+D	$IFDEF einfügen
Umsch+Strg+C	Kodevervollständigung (Class Completion)

Tabelle 1.2: IDE - Tastenkombinationen Bearbeiten

`Version: $LastChangedRevision: 38 $` [2]

[1] /Lazarus_IDE_Tools/de#.C3.9Cbersichtstabelle_der_IDE_Tastenkombinationen

[2] Autor: Andreas Frieß
Lizenz: GFDL

Suchen	
Tastenkürzel	Erklärung
F3	nächstes suchen
Umsch+F3	vorheriges suchen
Strg+R	Ersetzen
Strg+E	Inkremtielle Suche
Strg+G	Zu Zeile springen
Strg+H	Zurückspringen
Umsch+Strg+H	Vorwärtsspringen
Strg+F8	Zum nächsten Fehler springen
Umsch+Strg+F8	Zum vorherigen Fehler springen
Alt+Up	Deklaration unter Cursor suchen
Strg+Enter	Datei unter Cursor öffnen
Umsch+Strg+G	Prozedur Liste

Tabelle 1.3: IDE - Tastenkombinationen Suchen

Ansicht	
Tastenkürzel	Erklärung
F11	Objektinspektor anzeigen
Strg+F12	Unitliste anzeigen
Umsch+F12	Formularliste anzeigen
F12	Formulare/Unit Anzeige umschalten
Strg+Alt+F	Suchergebnisse anzeigen
Strg+Alt+B	Überwachte Ausdrücke
Strg+Alt+W	Haltepunkte
Strg+Alt+L	Lokale Variablen
Strg+Alt+S	Aufruf Stack

Tabelle 1.4: IDE - TastenkombinationenAnsicht

Projekt	
Tastenkürzel	Erklärung
Strg+F11	Projekt öffnen
Umsch+Strg+F11	Projekt Einstelllungen
Umsch+F11	Datei ins Projekt übernehmen

Tabelle 1.5: IDE - Tastenkombinationen Projekt

Start	
Tastenkürzel	Erklärung
Strg+F9	Erstellen
F9	Start
F7	Ein Schritt
F4	Start bis Cursor
Strg+F2	Halt
Strg+F7	Prüfen/ändern
Strg+F5	Überwachung hinzufügen

Tabelle 1.6: IDE - Tastenkombinationen Start

Divers	
Tastenkürzel	Erklärung
Strg+Click	springt zur Deklaration eines Typen oder Variablen
Strg+Shift+Upe	schaltet zwischen Definition und Rumpf um
Strg+J	Code-Schablonen
Strg+Space	Bezeichnervervollständigung
Strg+W	Word Completion
Strg+Shift+Space	Parameter Hinweise

Tabelle 1.7: IDE - Tastenkombinationen

1.3 Variablen, Konstanten, Datentypen

Ein großer Teil der hier verwendeten Definitionen und Texte stammen aus dem 'Language Reference Guide[FPCLangRef]'. Es handelt sich hier nicht um eine exakte Übersetzung, sondern um eine für dieses Buch angepasste Form.

1.3.1 Konstanten

Bei Konstanten muß der Kompiler zur Kompilierzeit fähig sein, den Term aufzulösen. Das heisst das sehr viele Funktionen die nur zur Laufzeit zur Verfügung stehen nicht verwendet werden können. Die folgenden Operatoren können immer verwendet werden „- *+, -, *, /, not, and, or, div, mod, ord, chr, sizeof, pi, int, trunc, round, frac, odd*“.

Normale Konstanten

Die Deklaration von Konstanten ist nur für *Ordinale-, Reale-, Char-* und *Stringtypen* erlaubt.

```
const
  co_real = 3.14; { Reale Typen Konstante }
  co_int  = 25; { Integer Typen Konstante }
  co_char = 'A'; { Character Typen Konstante }
  co_str = 'Irgend ein String'; {String Typen Konstante}
```

```
  co_ls = SizeOf(Longint);
  co_ls_real = SizeOf(co_real);
```

Eine Zuweisung[3] an die Konstante ist nicht möglich.

```
  co_str := 'Ein anderer String';
```

Das ist bei normalen Konstanten nicht erlaubt und wird mit der Fehlermeldung *xxx.pas(47,11) Error: Variable identifier expected* quittiert. Die Fehlermeldung resultiert daher, das zwar eine Konstante mit dem Namen existiert, aber keine Variable.

Typisierte Konstanten

Typisierte Konstanten sind ein Weg um das Programm mit initialisierten Variablen zu versorgen. Im Gegensatz zu normalen Variablen, wo der Programmiere sich um die Initialisierung selbst kümmern muß, wird ihr Wert initialisiert wenn das Programm startet. Anders als bei normalen Konstanten kann ihnen zur Laufzeit neue Werte zugewiesen werden.

```
const
  co_char : char = 'A';{Character Typen Konstante}
  co_str : string = 'Irgendein String';{String Typen Konstante}
```

Somit ist das gültiger Code.

```
  co_str := 'Ein anderer String';
```

Typisierte Konstanten werden oft benutzt um Arrays und Records zu initialisieren.

```
const
  co_tt : array [1..3] of string[20] = ('Mike', 'Ros', 'Heh');
  co_ti : array [1..3] of Longint = (1,2,3);
```

Bei Arrays müssen die initialen Werte in runden Klammern angegeben werden, mit Kommas getrennt und die Anzahl der Elemente muß genau gleich sein, als die Anzahl der Elemente in der Deklaration.

```
type
  Point = record
    X,Y : Real
  end;

const
  aOrigin : Point = (X:0.0; Y:0.0);
```

In typisierten Recordkonstanten, muß jedes Element des Records spezifiziert sein und zwar in der Form: Feldname, Doppelpunkt, Werte getrennt durch Strichpunkt und das ganze von runden Klammern umschlossen. Die Felder müssen in der Reihenfolge in der Deklaration verwendet werden, ansonsten gibt es eine Kompiler Fehlermeldung *xxx.pas(47,11) Error: Some fields coming before "Y"weren't initialized*

[3] Nur mit typisierten Konstanten möglich

Resourcen Strings

Sie verhalten sich Grundlegend wie normale Konstanten, es aber nur der Typ Strings zugelassen. Zusätzlich sind sie nur im Modus *objfpc* und *Delphi* erlaubt. Resourcen Strings sind in erster Linie dazu da, um die Internationalisierung einfacher zu gestalten und einen einheitliche Weg für das behandeln von konstanten Strings zu haben.

Die Strings werden dazu in eigene Dateien abgelegt, auf die dann über ein Programmierinterface zur Laufzeit zugegriffen werden kann. Dadurch können Strings für die Übersetzung benutzt werden. Ausserdem kommen mit dem Free Pascal Kompiler Werkzeuge um die Resourcen Strings zu bearbeiten und in andere Formate (Linux, Windows, ...) zu konvertieren.

```
Resourcestring
  FileMenu = '&File...';
  EditMenu = '&Edit...';
```

1.3.2 Variablen

Begriff

Eine Variable ist eine genau bekannter Speicherbereich mit einem bestimmten Typ. Wenn Werte einer Variablen zugewiesen werden, so erzeugt der Free Pascal Compiler Maschinenkode um den Wert zu dem reservierten Speicherbereich zu transportieren. Wo sich die Variable befindet, ist vom Ort der Deklarierung abhängig.

Globale Variablen Sind Variablen die innerhalb einer Unit oder Programmes definiert sind, aber NICHT innerhalb von Prozeduren oder Funktionen. Diese werden auf fixen Speicherbereichen gelegt und sind während der ganzen Programmbearbeitung verfügbar.

Lokale Variablen Sind innerhalb von Prozeduren oder Funktionen definiert. Sie werden auf den Stack, daher nicht auf einen fixen Speicherbereichen gelegt.

Beim FPC und somit auch innerhalb von Lazarus ist das bereitstellen der Speicherbereiche komplett transparent[4]. Das Hantieren (Schreiben, Lesen) mit den Werten erfolgt ebenfalls transparent, kann aber explizit durch den Programmierer vorgegeben werden, durch das verwenden von Eigenschaften(Properties).

Variablen müssen extra deklariert werden, wenn sie gebraucht werden. Es wird solange kein Speicherbereich zur Verfügung gestellt, bis die Variable deklariert ist. Wird eine Variable verwendet die nicht zuerst deklariert ist, so wird vom Compiler eine Fehlermeldung erzeugt.

Deklaration

Hier ist Platz für Informationen zum Thema. Ich bin leider noch nicht soweit.

[4] Für den Programmierer nicht sichtbar, im Hintergrund

1.3.3 Initialisierung

Variablen, wie auch andere Indentifizierer gehorchen den generellen Regeln der Gültigkeit. Zusätzlich werden die zu inititialisierten Variabeln zur folgenden Zeit initialisiert.

Globale initialisierte Variablen Werden einmalig initialisiert wenn des Programm startet.

Lokale initialisierte Variablen Werden jedes mal initialisiert wenn die Funktion oder Prozedur ausgeführt wird.

Man beachte das das Verhalten von lokal initialisierten Variablen ein anderes ist, als von lokal definierten Konstanten. Eine lokal definierte Konstante verhält sich wie eine global initialisierte Variable.

Welche Variablen werden überhaupt, wie initialisiert ?

AnsiString teilweise Der Platz für den Pointer wird erstellt und mit *nil* vorbelegt. Gültig nur für globale und lokale AnsiStrings, die ein Teil einer Struktur sind (Statische Arrays, Records und Objekte).

Typisierte Konstanten Verhält sich wie eine Globale initialisierte Konstante, wird einmalig initialisiert wenn des Programm startet, auch wenn sie sich Lokal in einer Prozedur oder Funktion befindet.

Es werden die meisten Variablen aus Geschwindigkeitsgründen nicht initialisiert. Folgendes soll als Beispiel dienen.

```
var
  anArray : array[1..100000] of string;
```

Bei dieser Deklaration wird, wenn die Variable initialisiert wird, hunderttausend Mal der Wert *nil* in das Array geschrieben. Das braucht seine Zeit, auch wenn die noch so kurz ist. Befindet sich der Code in einer Prozedur oder Funktion, die in einer Schleife aufgerufen wird, so kann der Zeitaufwand dafür erheblich werden.

1.3.4 Datentypen

ToDo: Hier ist Platz für Informationen zum Thema. Ich bin leider noch nicht soweit.

`Version: $LastChangedRevision: $` [5]

[5] Originalautor: Michaël Van Canneyt, Lizenz: free
Übersetzung und Bearbeitung: Andreas Frieß

1.4 Datenbanken

1.4.1 Einführung in die Datenbanktheorie

Das Folgende Kapitel wendet sich speziell an den Personenkreis der in die Theorie von Datenbanken noch nicht so eingedrungen ist, beziehungsweise dient als Nachschlagwerk für die verschiedenen Begriffe. Man kann also bei entsprechenden Vorwissen die folgenden erklärenden Kapitel überspringen und bei Bedarf nachschlagen.

Begriffe

Um Missverständnisse auszuschließen und zu einer gemeinsamen Sprachregelung zu kommen, sollte man die verwendeten Begriffe möglichst genau definieren:

- Eine Datenmenge ist eine Menge von einzelnen Datensätzen. Jeder Datensatz besteht aus mindesten einem Feld. Die Herkunft dieser Datenmenge ist nicht festgelegt.
- Eine Tabelle ist als Datenbankbestandteil eine spezielle Ausführung einer Datenmenge. Die Tabelle speichert als physisches vorhandenes Element die Daten.
- Eine Abfrage ist eine virtuelle Datenmenge, die den Inhalt von tatsächlich vorhandenen Tabellen in einer frei wählbaren Anordnung abbildet, manchmal auch Projektion genannt.
- Eine Datenbank ist eine Zusammenstellung von logisch zusammengehörigen Tabellen.
- Ein Datenbankserver verwaltet verschiedene Datenbanken und stellt auch das Management, die Sicherung und andere Verwaltungsdienste zur Verfügung.

Wichtige Informationen bei der Entwicklung einer Datenbankanwendung sind das Verhalten der Datenbank bzw. des Datenbankservers selbst.

Was ist eine Datenbank

Eine Datenbank ist eine geordnete Sammlung von Daten, die auf irgendeine Weise miteinander in Beziehung stehen.

Die ersten Generationen von Datenbanken waren sogenannte File-Systeme. Zuerst auf Band, dann auch auf Festplatten. In diesem Datei-Systemen wurden die Daten nacheinander abgespeichert. Um auf einen bestimmten Datensatz zu zugreifen, muss man an den Anfang der Datei (oder Bandes) gehen und anschließend alle Datensätze durchlaufen, bis man den richtigen gefunden hat. Somit ist auch klar, das ein einfaches sortieren oder einfügen von Daten in eine sortierte Datenmenge enormen Aufwand und Kapazität erfordert hat. Um diesen Beschränkungen zu entfliehen, (und durch neue, schnellere und größere Festplatten ermöglicht) haben sich aus diesen System die heutigen relationalen oder objektorientierten Datenbanken entwickelt. Die derzeit am meisten verwendeten Datenbanken sind die relationalen und im weiteren werde ich nur noch diese behandeln.

Desktop und Client-Server Datenbankarten

Gerade der Begriff Netzwerkdatenbank ist sehr verwirrend. Wird damit eine Access-Datenbank, die mehrere Benutzer über das Netzwerk verwenden so bezeichnet ? Oder eine Serverbasierende Datenbank? Die folgenden Erklärungen sollten die Begriffe klarer werden lassen.

Stand-Alone Datenbank Eine Stand-Alone Datenbank ist eine Desktop-Datenbank, es befinden sich daher die Daten auf dem Arbeitsplatzrechner. Auf die Daten kann immer nur ein Anwender, mit immer nur einem Programm zugreifen. Es ist zwar prinzipiell möglich über ein Netzwerk auf die Datenbankdatei zuzugreifen, aber es kann der eine nur in der Datenbank arbeiten, wenn der andere sein Programm geschlossen hat. Probleme die durch den gleichzeitigen Zugriff entstehen können daher gar nicht auftreten. Bei jeder etwas umfangreicheren Datenbank wird dieses Verhalten zu einem Engpass. Man stelle sich nur vor, der eine Benutzer öffnet die Datenbankdatei und vergisst auf das schließen des Programms. Kein anderer Benutzer kann die Daten in der Zwischenzeit benutzen!

File-Share Datenbank Moderne Netzwerke bieten die Möglichkeit, dass mehrer Anwender auf ein und dieselbe Datei zugreifen. Auf diese Weise ist es auch möglich das mehrer Programme auf ein und dieselbe Datenbankdatei zugreifen. Diese Version der Desktop-Datenbank nennt man File-Share Datenbank und damit ist bereits ein echter Mehrbenutzer Betrieb möglich. Das ganze hat jedoch (unter anderem) einen entscheidenden Nachteil: Die Datenverarbeitung erfolgt auf den Arbeitsplatzrechnern. Für Abfragen muss der jeweilige ganze Datenbestand zu den Arbeitsplatzrechnern gebracht werden, dementsprechend hoch ist die Belastung für das Netzwerk. Weiter können auch Störungen am Netzwerk leicht zu Datenverlust bzw. zu inkonsistenz der Datenbeständen führen.

Client-Server Datenbank Bei Client-Server Datenbanken hat nur der Datenbankserver selbst direkten Zugriff auf die Dateien des Datenbestandes. Anfragen werden somit direkt an den Datenbankserver gestellt, von diesem bearbeitet und die Ergebnisse an den Arbeitsplatzrechner zurückgeliefert. Die Verwaltung beim gleichzeitigen Zugriff durch mehrer Arbeitsplatzrechner obliegt dem Datenbankserver. Falls eine Verbindung durch eine Störung abbricht, so wird dieses erkannt und die noch nicht kompletten Anfragen verworfen und somit die Konsistenz der Daten erhalten. Gerade zur File-Share Datenbank können Netzbelastungen drastisch gesenkt werden. Man stelle sich nur vor, das man den größten Wert aus einer unsortierten Tabelle mit 1 Millionen Datensätze habe will. Bei der File-Share Datenbank müssen alle Datensätze geholt und bearbeitet werden, bei der Client Server Datenbank nur das Ergebnis. Weitere Bereiche sind die Möglichkeit Backups zu erstellen während die Datenbank weiter in Verwendung ist.

Relationale Datenbanken

Der Begriff relationale Datenbank geht auf einen Artikel von E. F. Codd zurück, der 1970 veröffentlich wurde. Codd bezeichnet Datenbanken als "minimal relational", wenn sie folgende Bedingungen erfüllen. [list] [*]Die Informationen werden einheitlich in Form von Tabellen repräsentiert. [*]Der Anwender sieht keine Verweisstrukturen zwischen den Tabellen. [*]Es gibt

mindestens die Operation der Selektion, der Projektion und des JOIN definiert. [/list] 1985 veröffentlichte Codd zwölf Regeln, die relationalen Datenbanken im strengeren Sinn definieren, Ende 1990 veröffentlichte er ein Buch über relationale Datenbanken, in dem er die einstigen zwölf Regeln des relationalen Modells auf 333 Regeln differenziert. Dadurch wird die Relationalität einer Datenbank bis ins letzte Detail festgeschrieben Soweit die Theorie, normalerweise sprechen Hersteller von relationalen Datenbanken, wenn die Mehrheit der 12 Regeln eingehalten werden.

Begriffe in relationalen Datenbanken Man kann nicht über relationale Datenbanken sprechen, ohne zuvor einige Begriffe zu klären.

Relationen Eine Relation ist gleich einer Tabelle. Die Daten werden daher in Relationen gespeichert.

Attribut und Tuples Die Attribute sind die Spalten einer Relation (Tabelle), die Tuples sind die Datensätze.

Degree und Kardinalität Die Zahl der Attribute einer Relation nennt man Degree, das ist der Ausdehnungsgrad. Die Zahl der Tuples ist die Kardinalität. Eine Relation mit Degree Null macht keinen Sinn, eine Kardinalität von NULL Tuples hingegen ist eine leere Relation.

Domain Eine Domain ist ein Wertebereich. Man kann z.B. eine Domain Nachnamen vom Type „Zeichen mit Länge 20“ erstellen. Diese wird immer dann verwendet wenn man Datenspalten mit dem Type „Nachname“ erstellen muss. Warum dieser Umweg? Wenn man später Tabellen miteinander verknüpft (in Relation bringt), so müssen die Spalten (Attribute) der gleichen Domain unterliegen. Habe ich vorher eine Domain definiert, so gibt es keine Probleme. Weiter ist es kein Problem wenn man draufkommt das die „Nachnamen“ länger sind, so kann die Definition der Domain geändert werden und alle Spalten haben die richtige Länge. Würde ich beim händischen Editieren eine Spalte vergessen so würde die Datenbank nicht mehr korrekt arbeiten.

NULL Ein „Nichtwert“ der anfangs immer für Verwirrung sorgt ist NULL. NULL ist nicht gleich Null! Ein besserer Ausdruck wäre UNBEKANNT. NULL macht übrigens aus einer zweiwertigen Logik (Ja/Nein) eine dreiwertige (Ja/Nein/Unbekannt). Vorstellen kann man es sich am besten mit einem Beispiel. Jedem Konferenzraum kann ein Beamer mit seiner Nummer zugeteilt werden. Die Räume welche keinen Beamer besitzen (zuwenige Beamer vorhanden) bekommen als Wert NULL zugeteilt, da ja ein nicht vorhandener Beamer auch keine Nummer besitzt, folglich also unbekannt ist.

Schlüssel Im Zuge der Normalisierung (welche später erklärt wird) werden die Daten auf viele Tabellen vereilt. Um dieses Tabellen wieder richtig in Relation zu bringen werden verschiedene Schlüssel, auch Keys genannt, verwendet.

Primärschlüssel (Primary Key) Jede Relation (Tabelle) besitzt einen Primärschlüssel um einen Datensatz eindeutig zu identifizieren. Ausnahmen von dieser Regel gibt es nur bei „M:N Verknüpfungen“ für die Zwischentabellen verwendet werden (die meisten Datenbanksysteme können diese Verknüpfungen nicht direkt abbilden). Ein Primärschlüssel ist immer eindeutig und ohne Duplikate. Meistens wird dafür eine fortlaufende Nummer verwendet, ist aber nicht zwingend. Vorsicht bei bestimmten Fällen, denn selbst Sozialversicherungsnummern müssen nicht eindeutig sein !

Sekundärschlüssel (Secundary Keys) Werden dafür verwendet um bei bestimmten Datenbankoperationen die Effizienz zu steigern, da die Datensätze intern nicht ungeordnet sondern in sortierter Reihenfolge verwaltet werden. Beim verwenden sollte aber immer auf die zugrunde liegende Datenbank Rücksicht genommen werden, da die Vor- und Nachteile stark datenbankabhängig sind.

Fremdschlüssel (Foreign Key) Ist der Verweis in der Tabelle auf einen Primärschlüssel in einer anderen Tabelle. Gerade in relationalen Datenbanken gibt es sehr viele Verknüpfungen die auf Primär- und Fremdschlüsselpaaren aufbauen. Wichtig ist, das Primär- und Fremdschlüssel der gleichen Domain unterliegen.

Referentielle Integrität Die referentielle Integrität stellt sicher das die Daten zueinander (über Primär- und Fremdschlüssel) glaubhaft bleiben. Ein einfügen, ändern oder löschen ist zu verweigern wenn dadurch die Datenintegrität verletzt würde. Man kann zum Beispiel keine Datensätze aus der Personentabelle löschen, solange in der Tabelle der Bestelllungen auf diese Personen verwiesen wird.

Normalisierung Unter der Normalisierung einer Datenbank, wird die Technik des logischen Datenbankdesigns bezeichnet. Es erfolgt meistens durch das Schrittweise optimieren der Datenbank zur Designzeit. Theoretiker haben insgesamt 5 Stufen der Normalisierung herausgearbeitet, wobei in der Praxis meist nur die ersten 3 Stufen verwendet werden.

Warum wird meistens nur bis zur 3. Normalform normalisiert? Bei der Anwendungsentwicklung besteht meistens nicht das Ziel, möglichst den exakten theoretische Grundlagen zu entsprechen, sondern eine möglichst effiziente Komplettlösung für das Problem zu erhalten. Dazu gehört natürlich auch, die Rücksicht auf das verwendete Datenbanksystem und die damit zu erzielenden Performance. Es leuchtet jedem ein, das die Aufteilung der Informationen auf viele Tabellen bei einer Auswertung zu schlechteren Ergebnissen führt. Somit kann es sein, das in Teilbereichen sogar eine Denormalisierung aus Performancegründen nötig ist, oder der gewonnene Platz bzw. das Geschäftsmodell keine Normalisierung sinnvoll erscheinen lassen.

Grunddaten

Mit Grunddaten werden die Informationen bezeichnet, die Voraussetzung für die tägliche Arbeit sind und während des täglichen Betriebs anfallen. Sie stellen die Basis des Datenbanksystems dar. Die Grunddaten werden in zwei Kategorien, Stammdaten und Bewegungsdaten, eingeteilt.

Ausgangslage	Alle Informationen in einer Tabelle
1. Normalform	Jede Spalte einer Tabelle enthält unteilbare Informationen. Die Datensätze verwenden keine sich wiederholenden Informationen, die nicht auch zu einer separaten Gruppe zusammengefasst werden könnten
2 Normalform	Es wird die 1. Normalform eingehalten und alle Informationen in nicht Schlüsselfeldern hängen nur vom kompletten Primärschlüssel ab
3 Normalform	Es wird die 2 Normalform eingehalten und alle Informationen in den nicht Schlüsselfeldern sind untereinander nicht abhängig.
4 Normalform	Es wird die 3. Normalform eingehalten und in gleichen Tabellen sind keine unabhängigen Objekte vorhanden, zwischen denen eine m:n Beziehung bestehen könnte
5 Normalform	Die normalisierte Datenbank kann nicht weiter in Tabellen mit weniger Attributen konvertiert werden. Es muss sich jederzeit der unnormalisierte Ursprungszustand ohne Informationsverlust herstellen lassen

Tabelle 1.8: Datenbank Normalisierungstufen Übersicht

Stammdaten Stammdaten sind diejenigen Grunddaten, die über einen längeren Zeitraum benötigt werden. Sie bilden den Grundbestand an Daten für das Datenbanksystem und werden auch als Bestandsdaten bezeichnet. Stammdaten weisen eine geringe Änderungshäufigkeit auf. Üblicherweise ist bei der Neuanlage von Stammdaten noch nicht bekannt, wann Änderungen zu erwarten und wie lange die Daten insgesamt gültig sind. Da auf Stammdaten häufig zugegriffen wird, ist ihre aktuelle Pflege notwendig, so dass Änderungen unmittelbar im Datenbestand nachgezogen werden sollten. Somit ist auch die normale Zugriffsart festgelegt, auf Stammdaten wir meistens in Verbindung mit Abfragen lesend zugegriffen, nur die Wartung erfolgt schreibend.

Bewegungsdaten Im Gegensatz zu Stammdaten haben Bewegungsdaten eine begrenzte Lebensdauer, die durch einen vorgegebenen Lebenszyklus beschrieben ist. Bewegungsdaten haben einen konkreten Zeitbezug, der für die Bedeutung und Interpretation der Information wichtig ist. Des weiteren beziehen sich Bewegungsdaten auf Stammdaten, weshalb sie auch als abgeleitete Daten bezeichnet werden. Da Bewegungsdaten gegenüber Stammdaten in der Menge mächtiger sind, ist gerade hier auf ein gutes Design zu achten. Betrachten wir es am Beispiel eines Zählers einer Station: Die Zähler werden im 10 Minutenrhythmus in die Bewegungsdaten eingefügt, das sind 144 Datensätze pro Tag, währenddessen der Stammdatenteil gleich bleibt, nämlich 1 Datensatz.

`Version: $LastChangedRevision: 55 $` [6]

[6] Autor: Andreas Frieß
Lizenz: GFDL

1.4.2 DDL Datendefinitionssprache

Die Datendefinitionssprache (Data Definition Language = DDL[7]) umfasst die Sprachteile von Datenbanksprache, mit deren Hilfe man Datenstrukturen wie Tabellen und andere ähnliche Elemente erzeugt. Es sind das die Befehle die mit *CREATE* beginnen, zum Beispiel *CREATE TABLE* und *CREATE INDEX*.

1.4.3 DML Datenveränderungssprache

Die Datenveränderungssprache (Data Manipulation Language = DML[8]) umfasst die Sprachteile von Datenbanksprache, die sich mit dem Lesen, Ändern und Löschen von Daten beschäftigen. Es sind das die Befehle *SELECT*, *INSERT*, *UPDATE* und *DELETE*.

Im folgend sehen wir uns die wichtigsten Befehle an, wobei ein Befehl besonders mächtig ist—*SELECT*—.

SELECT

SELECT ist einer der Vielseitigsten und am meisten eingesetzten Befehle überhaupt. Er dient zum Abfragen von Datenmengen aus der Datenbank. Er ermöglicht die Abfrage von Daten aus den Tabellen (die Projektion)

Einfachste Darstellung Bei der einfachsten Abfrage haben wir es mit wenigen Schlüsselwörter zu tun.

```
SELECT [DISTINCT] 'Auswahlliste'
  FROM 'Quelle'
  WHERE 'Where-Klausel'
  [GROUP BY ('Group-by-Attribut')+
  [HAVING 'Having-Klausel']]
  [ORDER BY (''Sortierungsattribut'' [ASC|DESC])+];
```

Mit SELECT wird die Abfrage eingeleitet, anschliessend kommt die Auswahlliste der gewünschten Tabellenspalten. Dann das FROM , das angibt welche Tabellen die Daten bereitstellen und das WHERE, das angibt nach welchen Kriterien die Daten vorselektiert werden.

SELECT 'Auswahlliste' Hier gibt man die einzelnen Tabellenspalten an, die sich später in der rückgelieferten Datenmenge finden. Sind die Name nicht eindeutig genug, besonders wenn mehrere Tabellen betroffen sind, so muß man den Tabellennamen und eventuell auch die Datenbank angeben, sprich die Eindeutigkeit qualifizieren. Somit kann ein SELECT so aussehen: `SELECT␣MyDB.STPerson.Vorname`, meistens wird die verkürzte Schreibweise verwendet wie `SELECT␣Vorname`.

In komplexeren Abfragen kann auch statt dem Tabellnamen alleine, eine Funktion stehen. Damit kann man in Datenmengen mathematische und statistische Funktionen verwenden. Auch

[7] siehe auch http://de.wikipedia.org/wiki/Data_Definition_Language
[8] siehe auch http://de.wikipedia.org/wiki/Data_Manipulation_Language

Verzweigungen und Berechnungen sind möglich. Zu diesen Punkten kommen dann an späterer Stelle die entsprechden Erklärungen.

DISTINCT erzwingt die Vermeidung von doppelten Datensätzen. Es überspringt in der Ausgabe die mehrfach vorkommenden Datensätze. Die Folge ist ein höherer Aufwand am Server, da das urspüngliche Ergebnis (ohne DISTINCT) meist noch entsprechend nachbearbeitet werden muß. Bevor man DISTINCT zur Korrektur unerklärlicher doppelter Datensätze verwendet (Verschleiern von Problemen), sollte man sich zuerst seine JOINS und WHERE Klauseln ganz genau ansehen, denn dort liegt meistens das Problem.

Eine gerne verwendete Abkürzung stellt das beliebte `SELECT␣*` dar. Hier darf dann das Programm zur Laufzeit erraten, welche Spalten es gibt und von welchen Typ sie sind. Es einfach zu verwenden, birgt aber in fertigen Programmen einiges an versteckten Fehlerquellen. Wird die dem SELECT zugrunde liegende Tabelle geändert oder auch nur Spalten getauscht, so kann es sein, daß Fehler nicht beim ausführen des Statements auffallen (Spalte xx nicht gefunden) und erst andere Programmteile dann unklare Fehlermeldung produzieren. Ich empfehle deshalb, die Spaltennamen wirklich anzugeben. Es gibt Programme die gerade diese Eigenschaft ausnutzen, dort wird es aber bewusst gemacht und entsprechend abgesichert.

FROM 'Quelle' FROM gibt die Quelle der Daten an. Es ist einfach der Tabellenname, eine Verbindung von Tabellen (JOIN) oder auch eine untergelagerte SELECT Anweisung.

WHERE 'Where-Klausel' Die WHERE - Klausel schränkt die Ergebnisdatenmenge ein. Bei einfachen Beispielen wird sie gerne, oft zu Unrecht, weggelassen. Wir müssen aber immer Datenmengen betrachten, das Ergebnis kann eine Zeile oder eine million Zeilen sein. Wenn wir in einer Adressdatenbank eine Personen suchen, so wird es trotzdem sinnvoll sein, von Haus aus die Suche einzuschränken. Denn alles was nicht an Daten sinnlos übertragen werden muß, spart Bandbreite, Speicher und erhöht die Geschwindigkeit.

GROUP BY ('Group-by-Attribut' Die Daten werden nach dem Group-by-Attributen zusammengefasst (gruppiert). Dazu müssen auch in der Auswahlliste, für alle nicht durch GROUP BY erfassten Spalten, entsprechende Operationen verwendet werden (SUM, AVG, MIN, ...).

HAVING 'Having-Klausel' Ist als ein WHERE zu betrachten, das allerdings erst **nach** dem Gruppieren wirksam ist und deshalb nur auf die Gruppierungsfunktionen wirksam ist.

ORDER BY ("Sortierungsattribut" [ASC|DESC]) Die ausgegeben Daten werden entsprechend sortiert. Das Sortierungsattribut sind die entsprechden Spaltennamen in der reihenfolge wie sortiert werden soll.

Beispiele zu SELECT

*SELECT * FROM st_person;*
Ohne Einschränkung oder Sortierung.

```
"STPerson","cVName","cFName","cMName","cRName"
378,"Hope","Giordano","HoGi","Hope0Giordano"
379,"Rose","Bruno","RoBr","Rose4Bruno"
380,"Lauren","Morgan","LaMo","Lauren4Morgan"
381,"Megan","Coleman","MeCo","Megan9Coleman"
```

*SELECT * FROM st_person where STPerson = 875;*
Die Person deren ID 875 ist.

```
"STPerson","cVName","cFName","cMName","cRName"
875,"Hannah","Collins","HaCo","Hannah3Collins"
```

*SELECT * FROM st_person where cVName like 'H%';*
Alle Personen deren Vorname mit „H“ beginnt. Das Prozentzeichen „%“ ist eine Wildcard. So wie bei manchen Betriebssystemen der Stern „*“.

```
"STPerson","cVName","cFName","cMName","cRName"
875,"Hannah","Collins","HaCo","Hannah3Collins"
406,"Hannah","Cook","HaCo","Hannah9Cook"
845,"Hannah","Doyle","HaDo","Hannah9Doyle"
1363,"Hannah","Foster","HaFo","Hannah6Foster"
```

*SELECT * FROM st_person order by cFName;*
Alle Personen, aber nach Familienname aufsteigend sortiert.

```
"STPerson","cVName","cFName","cMName","cRName"
1258,"Caitlin","Adams","CaAd","Caitlin4Adams"
687,"Taylor","Alexander","TaAl","Taylor5Alexander"
644,"Renee","Alexander","ReAl","Renee8Alexander"
885,"Taylor","Alexander","TaAl","Taylor3Alexander"
1131,"Sarah","Alexander","SaAl","Sarah5Alexander"
603,"Megan","Allen","MeAl","Megan0Allen"
1172,"Isabella","Allen","IsAl","Isabella0Allen"
472,"Isabelle","Allen","IsAl","Isabelle6Allen"
```

*SELECT * FROM st_person order by cFName desc;*
Alle Personen, aber nach Familienname absteigend sortiert.

```
"STPerson","cVName","cFName","cMName","cRName"
842,"Isabella","Young","IsYo","Isabella6Young"
1232,"Taylor","Young","TaYo","Taylor3Young"
427,"Ashley","Young","AsYo","Ashley3Young"
420,"Kyla","Young","KyYo","Kyla2Young"
668,"Alexandra","Wright","AlWr","Alexandra6Wright"
1070,"Madison","Wright","MaWr","Madison5Wright"
```

*SELECT * FROM st_person where cVName like 'H%' order by cFName;*
Alle Personen deren Vorname mit „H“ beginnt und nach Familienname aufsteigend sortiert.

```
"STPerson","cVName","cFName","cMName","cRName"
932,"Hope","Bell","HoBe","Hope3Bell"
1194,"Harmony","Campbell","HaCa","Harmony7Campbell"
515,"Harmony","Clark","HaCl","Harmony4Clark"
1279,"Holly","Collins","HoCo","Holly7Collins"
875,"Hannah","Collins","HaCo","Hannah3Collins"
406,"Hannah","Cook","HaCo","Hannah9Cook"
1296,"Harmony","Diaz","HaDi","Harmony1Diaz"
```

SELECT cVName, cFName FROM st_person where cVName like 'H%' order by cFName; Anzeige nur der Spalten „cVName“ und „cFName“ und alle Personen deren Vorname mit „H“ beginnt und nach Familienname aufsteigend sortiert.

```
"cVName","cFName"
"Hope","Bell"
"Harmony","Campbell"
"Harmony","Clark"
"Holly","Collins"
"Hannah","Collins"
"Hannah","Cook"
"Harmony","Diaz"
```

SELECT cVName, cFName FROM st_person where (cVName like 'H%') or (cVName like 'A%') order by cFName;
Anzeige nur der Spalten „cVName“ und „cFName“ und alle Personen deren Vorname mit „H“ oder „A“ beginnt und nach Familienname aufsteigend sortiert.

```
"Alyssa","Butler"
"Abby","Butler"
"Ashley","Butler"
"Ashley","Byrne"
"Harmony","Campbell"
"Harmony","Clark"
"Anna","Clark"
"Abby","Coleman"
"Hannah","Collins"
"Amelia","Collins"
"Anna","Collins"
```

SELECT count(cVName), cVName FROM st_person group by cVName order by count(cVName) desc; Anzahl der Vorkommen der Vornamen absteigend sortiert.

```
"count(cVName)","cVName"
19,"Amelia"
19,"Angel"
17,"Laura"
```

```
16,"Elizabeth"
16,"Paris"
16,"Ruby"
15,"Caitlin"
14,"Sophie"
14,"Abby"
14,"Mikayla"
```

SELECT count(cVName) as Anzahl, cVName as Vorname FROM st_person group by cVName order by count(cVName) desc; Anzahl der Vorkommen der Vornamen absteigend sortiert. Dazu noch die Spaltennamen geändert.

```
"Anzahl","Vorname"
19,"Amelia"
19,"Angel"
17,"Laura"
16,"Elizabeth"
16,"Paris"
16,"Ruby"
15,"Caitlin"
14,"Sophie"
14,"Abby"
14,"Mikayla"
```

SELECT count(cVName) as Anzahl, cVName as Vorname FROM st_person group by cVName desc having Anzahl = 16; Anzahl der Vorkommen der Vornamen, Dazu noch die Spaltennamen geändert und die Anzahl muß sechzehn sein

```
"Anzahl","Vorname"
16,"Elizabeth"
16,"Paris"
16,"Ruby"
```

INSERT

INSERT wird für das Einfügen von Datensätzen in eine Tabelle verwendet.

Einfache Darstellung Beim einfachen Einfügen haben wir es mit wenigen Schlüsselwörter zu tun.

```
INSERT 'Ziel' ('Spaltenliste')
  VALUES ('Werteliste')
```

Bei dem Ziel handelt es sich um die Tabelle in die die Daten eingefügt werden sollen. Die Spaltenliste kann auch weggelassen werden, wenn die Werteliste in der exakt gleichen Reihenfolge ist, wie die Definition in der Tabelle.

Wenn die Spaltenliste vorhanden ist, so müssen die Werte in der Werteliste in der gleichen Reihenfolge stehen. Es muß dann aber nicht die Reihenfolge und Anzahl der Spalten mit der Tabellen Definition übereinstimmen. Das wird normalerweise verwendet, da an Spalten mit automatischen Zählern (meist Primärschlüssel) keine Werte übergeben werden dürfen!

Beispiele zu INSERT

INSERT st_person (cVName,cFName,cMName, cRName) VALUES ("Hope","Giordano","HoGi", "Hope0Giordano"); Bei *INSERT* und anderen Befehlen zur Erstellung von Daten, gibt es kein Ergebnismenge.

```
SELECT * FROM st_person;

"STPerson","cVName","cFName","cMName","cRName"
1,"Hope","Giordano","HoGi","Hope0Giordano"
2,"Harmony","Campbell","HaCa","Harmony7Campbell"
3,"Harmony","Clark","HaCl","Harmony4Clark"
4,"Holly","Collins","HoCo","Holly7Collins"
```

Die Spalte STPerson, ist zwar beim INSERT nicht angegeben, wird vom System automatisch vergeben.

Ein weiteres Beispiel befindet unter Projekt MySQLTestData

UPDATE

Mit *UPDATE* können die Daten in den Tabellen geändert werden.

Einfache Darstellung Beim einfachen Ändern haben wir es mit wenigen Schlüsselwörter zu tun.

```
UPDATE 'Ziel' SET 'Spalte1' = 'Wert1' , 'Spalte2' = 'Wert2'
  WHERE 'Where-Klausel'
```

Bei dem Ziel handelt es sich um die Tabelle in der die Daten geändert werden sollen. Nach dem Schlüsselwort *SET* erfolgt die Auflistung der Spalten mit den neuen Wert.

Wichtig ist hier die *WHERE*-Klausel! Fehlt sie so werden **alle Datensätze** der Tabelle geändert! Soll nur ein einziges Datensatz von der Änderung geändert werden, so muß die Einschränkung richtig definiert sein. Normalerweise wird hier der Primärschlüssel (auch zusammengesetzt) verwendet, denn so ist die Eindeutigkeit sichergestellt. Bezüglich der Möglichkeiten der *WHERE*-Klausel bitte beim Befehl *SELECT* nachzulesen.

Beispiele zu UPDATE

UPDATE st_person set cRName = "HoGi"WHERE STPerson = 1; Bei *UPDATE* und anderen Befehlen zur Erstellung von Daten, gibt es kein Ergebnismenge.

```
SELECT * FROM st_person;

"STPerson","cVName","cFName","cMName","cRName"
1,"Hope","Giordano","HoGi","HoGi"
2,"Harmony","Campbell","HaCa","Harmony7Campbell"
3,"Harmony","Clark","HaCl","Harmony4Clark"
4,"Holly","Collins","HoCo","Holly7Collins"
```

UPDATE st_person set cVName = "Hopper", cRName = "HoGi1"WHERE STPerson = 1; Hier werden zwei Spalten gleichzeitig geändert

```
SELECT * FROM st_person;

"STPerson","cVName","cFName","cMName","cRName"
1,"Hopper","Giordano","HoGi","HoGi1"
2,"Harmony","Campbell","HaCa","Harmony7Campbell"
3,"Harmony","Clark","HaCl","Harmony4Clark"
4,"Holly","Collins","HoCo","Holly7Collins"
```

UPDATE st_person set cVName = "HaHa"WHERE cVName like „Ha“; Hier werden mehrere Datensätze gleichzeitig geändert

```
SELECT * FROM st_person;

"STPerson","cVName","cFName","cMName","cRName"
1,"Hopper","Giordano","HoGi","HoGi1"
2,"HaHa","Campbell","HaCa","Harmony7Campbell"
3,"HaHa","Clark","HaCl","Harmony4Clark"
4,"Holly","Collins","HoCo","Holly7Collins"
```

DELETE

Mittels dem Befehl *DELETE*werden Datensätze in der Datenbank gelöscht.

Einfache Darstellung Beim einfachsten DELETE haben wir es mit wenigen Schlüsselwörter zu tun.

```
DELETE 'Ziel'
  WHERE 'Where-Klausel'
```

Aber Vorsicht, ohne entsprechende *WHERE*-Klausel werden alle betroffenen Datensätze gelöscht. Daher gilt, fehlt die WHERE-Klausel, so werden **alle Datensätze** unwiderruflich gelöscht!

Beispiele zu DELETE

DELETE FROM st_person WHERE STPerson = 1; Bei *DELETE* und anderen Befehlen zur Erstellung von Daten, gibt es kein Ergebnismenge. Hier wird der Datensatz mit dem Wert 1 in der Spalte STPerson gelöscht.

```
SELECT * FROM st_person;

"STPerson","cVName","cFName","cMName","cRName"
1,"Hope","Giordano","HoGi","HoGi"
2,"Harmony","Campbell","HaCa","Harmony7Campbell"
3,"Harmony","Clark","HaCl","Harmony4Clark"
4,"Holly","Collins","HoCo","Holly7Collins"

DELETE FROM st_person WHERE STPerson = 1;
SELECT * FROM st_person;

"STPerson","cVName","cFName","cMName","cRName"
2,"Harmony","Campbell","HaCa","Harmony7Campbell"
3,"Harmony","Clark","HaCl","Harmony4Clark"
4,"Holly","Collins","HoCo","Holly7Collins"
```

DELETE st_person WHERE STPerson = 1; Bei *DELETE* und anderen Befehlen zur Erstellung von Daten, gibt es kein Ergebnismenge. Hier werden **alle** Datensätze in der Tabelle st_person gelöscht!

```
SELECT * FROM st_person;

"STPerson","cVName","cFName","cMName","cRName"
1,"Hope","Giordano","HoGi","HoGi"
2,"Harmony","Campbell","HaCa","Harmony7Campbell"
3,"Harmony","Clark","HaCl","Harmony4Clark"
4,"Holly","Collins","HoCo","Holly7Collins"

DELETE FROM st_person;
SELECT * FROM st_person;

"STPerson","cVName","cFName","cMName","cRName"
```

1.4.4 DCL Datenkontrollsprache

Die Datenkontrollsprache (Data Control Language = DCL[9]) umfasst die Sprachteile von Datenbanksprache, mit deren Hilfe man die Rechte vergibt, Wartungen durchführt. Es sind das Befehle wie *GRANT* und *REVOKE*.

[9] siehe auch http://de.wikipedia.org/wiki/Data_Control_Language

```
Version: $LastChangedRevision: 52 $
```
[10]

[10] Autor: Andreas Frieß
Lizenz: GFDL

2 Codebeispiele

2.1 Objekte

2.1.1 Was ist ein Objekt

Was ist ein Objekt, eine gute Frage. Nähern wir uns einmal dem Begriff indem wir ein Objekt beschreiben. Ein Objekt hat **Eigenschaften** und kann **Aktionen** durchführen, weiters beinhaltet es Daten zum **Verwalten seiner Zustände**. Außerdem muß das Objekt auch einen **Namen** besitzen. Wie definieren wir ein Objekt? Unter (Lazarus) Pascal verwenden wir dazu die Klassendefinition *class*.

```
TMyObject = class
end;
```

Mit diesen Zeilen kann mal ein grundlegendes Objekt definieren. Es hat einen Namen, aber kann weder Aktionen durchführen, noch hat es Eigenschaften, aber es ist das minimalste Beispiel. In unseren Fall hat das Objekt den Namen „TMyObject“.

Sehen wir uns jetzt eine volle Definition an.

```
{ TMyGreatObject }
TMyGreatObject = class(TParentObject)
private
  FIsValid: Boolean;
  procedure SetIsValid(const AValue: Boolean);
protected
  function GetWert: integer;
  procedure SetWert(const AValue: integer);
public
  constructor Create; override;
  destructor Destroy; override;
published
  property IsValid : Boolean read FIsValid write SetIsValid;
  property Wert : integer read GetWert write SetWert;
end;

var
  aMyGreatObject : TMyGreatObject;
```

Mittels der Klassendefinition `class(TParentObject)` definieren wir ein Objekt mit dem Namen *TMyGreatObject*, das alle Eigenschaften, Aktionen etc. vom Elternobjekt erbt. Das heißt

alles was das Elternobjekt kann oder hat, hat unser Objekt auch. Durch die Deklarationen *private, protected, public und published* wird die Sichtbarkeit der Eigenschaften und Aktionen definiert.

private Nur hier in dieser Deklaration kann auf diese Eigenschaften und Aktionen zugegriffen werden

protected Wie private, zusätzlich kann auch bei Kindobjekten auf die Eigenschaften und Aktionen zugegriffen werden

public Auf die Eigenschaften und Aktionen kann immer zugegriffen werden

published Wie *public* zusätzlich sind die Eigenschaften auch im Objektinspektor verfügbar

Im *private* Teil ist eine Boolsche-Variable und eine Prozedur zum Ändern der Variablen definiert. Generell ändert man den Wert einer Variablen nicht direkt, sondern verwendet eine Prozedur dazu, in der auch eventuell die nötigen Prüfungen oder Bearbeitungsschritte durchgeführt werden. Als Namen für die Prozeduren[1] und Funktionen[2] die interne Variablen ändern haben sich *GetXXXX*[3] und *SetXXXX* durchgesetzt. Das setzt sich auch im *protected* Abschnitt fort.

Im *public* Bereich sind die Konstruktoren (immer *Create*[4]), Destruktoren (immer *Destroy*) und die von ausserhalb sichtbaren Variablen[5], Eigenschaften, Prozeduren und Funktionen.

2.1.2 Zur Laufzeit erzeugen und zerstören

Gerade am Anfang erweist sich das richtige Erzeugen und Löschen von Objekten als Problem. Denn die Zuweisung mittels *var aListe : TStringList;* reicht nicht aus. Damit wird nur der Platz für den Verweis auf das Objekt erzeugt, nicht aber das Objekt selbst! Zu diesem Zeitpunkt ist das Objekt, in diesem Fall eine Stringliste nicht in einem gebrauchsfähigen Zustand, der Verweis ist undefiniert (und zeigt ins Nirgendwo). Deshalb wird auch eine Anweisung wie *aListe.Create* nicht funktionieren und eine Exception auslösen.

Wie geht es also richtig. Alle Objekte haben einen Konstruktor, entweder direkt in der Klasse oder in einem Vorfahren, der aufgerufen werden kann und der das Objekt richtig erstellt und uns den richtigen Verweis zurück liefert, den wir dann in unserer Variablen ablegen können.

```
procedure TForm1.buObjektClick(Sender: TObject);
  var aListe : TStringList;
begin
  aListe := TStringList.Create;
  try
    aListe.Add('Eine Zeile');
    aListe.Add('Eine zweite Zeile');
    aListe.Add('Eine dritte Zeile');
```

[1] Zum Setzen
[2] Zum Abfragen
[3] Auch Getter und Setter genannt
[4] manchmal kommen Namensabwandlungen vor
[5] Variablen macht man normalerweise über Eigenschaften zugänglich

```
    Memo1.Lines.Assign(aListe);
  finally
    aListe.Free;
  end;
end;
```

Sehen wir uns den Beispielcode[6] einmal von oben nach unten an.

var aListe : TStringList; Definieren einer Variable für den Verweis auf ein Objekt von Typ *TStringlist*

aListe := TStringList.Create; Erzeugen des Objektes und initialisieren der Variablen mit dem Verweis

aListe.Add('Eine Zeile'); Einfügen eines Strings in die Stringliste

Memo1.Lines.Assign(aListe); Füllen des Memos mit den Strings aus der Stringliste aListe

aListe.Free; Freigeben des Objektes Stringliste durch aufruf von *Free*.

Nach dem Aufruf von *Free* ist der Inhalt der Variablen aListe nicht mehr definiert! Free ruft den Destruktor des Objektes auf. Der Inhalt zeigt jetzt auf kein gültiges Objekt mehr und enthält einen beliebigen Inhalt, er ist **nicht** Null oder nil.

Zusätzlich ist mit der Exceptionbehandlung *try..finally..end* sichergestellt, daß das Objekt sicher zerstört wird, auch wenn innerhalb des *try..finally* Blocks ein Fehler aufgetreten ist. Das zwischen dem *finally..end* eingegrenzte Anweisung wird in jedem Falle ausgeführt.

2.1.3 Was sollte man unterlassen

Es gibt bei Klassen immer wieder Kode, den man besser nicht schreiben sollte. Ich werde hier versuchen Beispiele zu bringen, meistens werden sie aus dem Forum stammen.

Erzeugung von Speicherleichen (Memoryleaks)

[7] Eine einfacher Weg um Speicherleichen zu erzeugen ist das 'wilde' zuweisen von Klassen. Meistens wird der Gedanke aus einer vermeintlichen Abkürzung im Kode geboren.

Bei der Deklaration ist ja noch alles in Ordnung.

```
Obj1 : TObject;
Obj2 : TObject;
```

Ôbj1,2 sind nur Zeiger die im Moment auf irgendwas zeigen wenn man drauf zugreifen will gibt es eine Zugriffsverletzung ('Access violation'), weil das Objekt ja noch nicht erstellt wurde.

```
Obj1 := TObject.Create;
Obj2 := TObject.Create;
```

[6] Aus dem Projekt: BspObjErzeug

[7] Basierend auf Informationen von christian[11] im deutschen Lazarusforum

jetzt zeigen unsere Zeiger auf die Objekte die erstellt wurden und die Objekte sind jetzt auch benutzbar.

Wenn man jetzt aus welchen Grund auch immer die folgende Zuweisung macht, hat man eine Leiche erschaffen.

```
Obj1 := Obj2;
```

Mit dieser Zuweisung zeigen beide Zeiger auf Obj2. Obj1 kann nie wieder benutzt werden, da jegliche Referenzierung darauf verschwunden ist. Somit kann Obj1 auch nicht mehr gelöscht und der Speicherplatz frei gegebnen werden, wie im folgenden Kode.

```
Obj1.Free;
Obj2.Free;
```

Das erste Obj1.Free wird anstandslos ausgeführt, da ja in Wirklichkeit das Obj2 freigegeben wird und nicht das ursprüngliche Obj1 - das ist ja in Vergessenheit geraten. Das nachfolgende Obj2.Free erzeugt jetzt eine Zugriffsverletzung, da das Objekt ja bereits freigegeben wurde und das ursprüngliche Obj1 ist eine Leiche (im Speicher-Keller).

```
Version: $LastChangedRevision: 59 $ [8]
```

[8] Autor: Andreas Frieß
Lizenz: GFDL

3 Bibliotheken

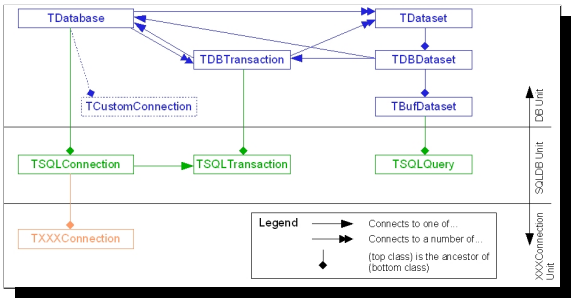
3.1 SQLdb

3.1.1 Beschreibung

Einleitung

SQLdb besteht im Wesentlichen aus Komponenten für die Verbindung (TxxxConnection), für das Verwalten von Transaktionen (TSQLTransaction) und dem Verwalten von Datenmengen (TSQLQuery).

Besonders die TSQLQuery ist eine mächtige Komponente, die einige automatismen eingebaut hat, die zwar das Leben erleichtern sollen, aber oft das Gegenteil bewirken können. In diesem Kapitel wollen wir uns die Komponenten einmal genauer ansehen. Das Bild aus der Wiki aus dem englischen Lazarusforum[LazEn][1] erklärt schön, wie die Komponenten zusammenhängen und in welchen Units sie sich befinden.



Debbuging von SQLdb

SQLdb ist ein Teil der FCL und nicht von Lazarus. Die FCL wird standardmässig nicht mit den für den Debugger notwendigen Informationen kompiliert, da der Kode so kompakter und kleiner ist. Wenn man für die Fehlersuche es anders benötigt, so muß man die fcl-db neu kompileren. Dazu muß man die kompletten FCL Sourcen haben, dann kann man in das Verzeichnis 'fpc/packages/fcl-db' gehen und mit 'make clean all OPT='-gl" das Paket neu kompiliren, anschliessen die neuen PPU's über die alten kopieren. Die Fehlersuche wird aber nur Personen empfohlen, die entsprechendes Wissen über die SQLdb Komponenten haben.

Active, Open oder ExecSQL

... das ist hier die Frage ? Um diese Frage zu beantworten, muß man sich vor Augen halten, was man von der Komponente will. Mittels dem Befehl Open fordert man eine Datenmenge, die Kardinalität[2] einer Datenmenge[3] kann auch Null sein, an. Mit ExecSQL wird nur eine Aktion

[1] *http://wiki.lazarus.freepascal.org/SQLdb_Programming_Reference*
[2] siehe Kapitel 1.4.1 auf Seite 17
[3] siehe Kapitel 1.4.1 auf Seite 15

angefordert ohne das eine Datenmenge zurück erwartet wird. Somit ist klar, das bei allen Daten liefernden Statements das Open zu verwenden ist. Welche Statements in SQL liefern überhaupt Datenmengen zurück ? Eigentlich gilt das nur für das 'SELECT' Statement, alle anderen ('INSERT', 'UPDATE', 'DELETE', ...) führen etwas aus, liefern aber keine Daten zurück. Ob man jetzt 'Open' verwendet oder 'Active:=true;' macht ist letztlich egal, 'Open' führt genau dieses Statement aus.

Eine Besonderheit ist die Behandlung von 'Stored Procedure' und 'Functions' auf SQL-Servern. Diese können eine Kombination im Verhalten darstellen. Dort ist dann ein ExecSQL angebracht und es können Datenmengen zurückgeliefert werden.

Zusammenfassung:

- Open, Active: Bei der Verwendung von 'SELECT'
- ExecSQL: Für alle anderen Statements

Wie kommen die geänderten Daten in die Datenbank

Eine änderung der Datenmenge alleine ist nicht ausreichend, um diese Änderung auch in der Datenbank sichtbar zu machen. Prinzipiell muß man jetzt zwei Wege unterscheiden. Einerseits kann man Änderungen im Zuge einer Transaktion in der Datenbank festschreiben durch das Abschliessen der Transaktion oder auch durch das dezitierte schreiben durch ApplyUpdates. Ich bin der Meinung, das man sich für einen der Wege entscheiden sollte, wenn man eine Datenmenge öffnet beziehungsweise anfordert. Arbeitet man mit Transdaktionen, dann soltte man ohne zwingenden Grund nicht mit ApplyUpdates das Transaktionsmanagment stören. Anderseits wenn man ohne explzite Transaktionen arbeitet, also Datenmengen öffnet ohne vorher Transaktionen geöffnet zu haben, dann darf man nicht vergessen die Änderungen entweder mittels ApplyUpdates zu übernehmen oder mittels CancelUpdates zu verwerfen. Vergisst man auf ApplyUpdates so ist dieses schlimmer als auf CancelUpdates zu vergessen. Denn ohne dem ApplyUpdates sind die Daten bei den meisten Datenbanken ganz einfach nicht in die Datenbank eingearbeitet und somit verloren.

Filtern, aber wo ?

Die Standardantwort ist im Stile von Radio Eriwan: 'Dort wo es sinnvoll ist'. Dazu muß man sich vor Augen halten, wo man eine Datenmenge überhaupt filtern kann. Dazu muß man unterscheiden in Desktop Datenbanken und Server Datenbanken. Bei Desktop Datenbanken kann die Frage schon obsolet sein, weil die Datenmenge sowieso nur lokal gefiltert werden kann. Bei Datenbankenservern schaut die Sachlage ganz anders aus. Denn die können Datenmengen sehr wohl, effizient vor verarbeiten und nur die wenigen Ergebnisse zurück transportieren. Damit wird am lokalen Rechner Netzwerkleistung, Speicher und Resourcen geschont. Somit kann man hier dem filtern am Server den Vorzug geben. Werden aber Daten erst am lokalen Rechner verknüpft, so kann man oft nur lokal filtern. Somit ist klar, das es stark auf das Design der Applikation an kommt, was sinnvoll ist.

Anzahl der Datensätze abfragen

Hier kann man generell zwei verschiedene Fälle unterscheiden. Einmal den Fall, das man wissen will, ob überhaupt Datensätze vorhanden sind und dem Fall, das man die Anzahl wissen will.

Generell ist es nur dann sinnvoll die Anzahl der Datensätze zu bestimmen, wenn eine Abfrage aktiv ist.

Ob Datensätze überhaupt vorhanden sind, kann man über die Abfrage von EOF[4] und BOF[5] machen. Sind beide vorhanden ('true') so muß die Datenmenge leer sein. Genau diese macht die Methode 'IsEmpty'. Somit kann man diese genau für diesen Fall verwenden.

Die Anzahl selbst der Datensätze, kann man theoretisch mittels der Eigenschaft 'RecordCount' abfragen. Alledings muß dazu auch der Datenbanktreiber[6] das unterstützen. Bis jetzt ist die Unterstützung auch nicht wirklich vorhanden. Weiters handelt es sich hier eher um eine Eigenschaft von Desktopdatenbanken, denn dort kann die Anzahl der Datensätze nicht anders festgestellt werden.

Die andere Variante die sich daher anbietet ist die SQL Abfrage selbst. So kann man mittels dem SQL-Statement `select count(row1) as Anzahl from table1 where ...` die Anzahl ermitteln.

Navigieren durch eine Datenmenge

Für das Navigieren durch die Datenmenge stehen ein paar Befehle zur Verfügung. Mit 'first' kommt man zum ersten, mit 'last' zum letzten, mit 'next' springt man auf den nächsten und mit mit 'prior' zum vorhergehenden Datensatz. Größere Bewegungen kann man mit 'MoveBy' machen. Das funktioniert vorwärts mit positiven Zahlen, rückwärts mit negativen Zahlen. Allerdings muß man bedenken, das ein 'MoveBy' nicht zwingend die volle Distanz verfahren kann, wenn die Grenzen der Datenmenge erreicht werden. Wenn also ein EOF oder BOF nach dem 'MoveBy' ansteht, so wird nicht die volle Distanz erreicht worden sein, man weiß aber nicht um wie viel verfahren wurde.

Was ist BOF und EOF

'BOF' bedeutet das man in der Datenmenge am Anfang, bei 'EOF' am Ende der Datenmenge steht. Wenn zum gleichen Zeitpunkt beide vorhanden sind, so ist das ein Zeichen, das die Datenmenge null ist.

Zugriff auf Felder

Auf die Felder[7] kann über die Eigenschaft 'Fields' zugegriffen werden. Zusätzlich kann über die Methode 'FieldByName' mittels des Feldnamens oder 'FieldByNumber' einfach auf die einzelnen Felder zugegriffen werden. Die Werte werden über die 'Values' Eigenschaft als variant zugewiesen oder über die entsprechenden 'AsInteger', 'AsString' und 'As.....'.

[4] End of File - Anfang der Daten
[5] Beginn of File - Ende der Daten
[6] Ist genaugenommen die Verbindungskomponente
[7] auch Attribute genannt

Zugriff auf Parameter

Auf die Felder der Parameter kann über die Eigenschaft 'Params' zugegriffen werden.Zusätzlich kann über die Methode 'ParamsByName' mittels des Feldnamens einfach auf die einzelnen Felder zugegriffen werden. Die Werte werden über die 'Values' Eigenschaft als variant zugewiesen oder über die entsprechenden 'AsInteger', 'AsString' und 'As.....'.

Im SQL-Statement werden die Parameter durch einen Doppelpunkt am Anfang des Names kenntlich gemacht. Zum Beispiel: `insert tablex (row1, row2) values (:param1, :param2)` Hier ist ':param1' einer der Parameter. Die Zuweisung im Programm erfolgt über `TQ1.Params.ParamByName('`

Schlüsselfelder

Als Schlüsselfelder werden die Felder[8] bezeichnet in dem der primäre Schlüssel der Tabelle gespeichert ist. Dieser sollte bei jeder änderbaren Datenmenge definiert sein, damit die Komponenente richtig die Datensätze ändern oder löschen kann. Schlüssel erzwingen eine Eindeutigkeit.

3.1.2 TxxxConnection

Genau genommen handelt es sich hier nicht nur um eine einfache Deklaration der Verbindung sondern um den lokalen Verwaltungsteil der Datenbank. Hier werden die gemeinsamen Methoden und EIgenschaften behandelt, im Folgenden die verschiedenen Connection mit den abweichenden Details behandelt.

Close

```
procedure Close;
```

Setzt ganz einfach die Eígenschaft active auf false.

Methode von TxxxConnection>TSQLConnection>TDatabase>TCustomConnection

EndTransaction

```
procedure EndTransaction; override;
```

Ruft die entsprechede Methode der Komponente Transaction auf.

Methode von TxxxConnection>TSQLConnection

ExecuteDirect

```
procedure ExecuteDirect(SQL : String); overload; virtual;

procedure ExecuteDirect(SQL : String; ATransaction : TSQLTransaction);
    overload; virtual;
```

Führt das SQL-Statement entweder im Kontext der default Transaktion oder mittels der Angegeben Transaktion aus.

[8] Schlüssel können über mehrere Felder gehen um eindeutig zu sein, oder weil sie zusammengesetzt sind

Methode von TxxxConnection>TSQLConnection

Parameter

SQL: string Das auszuführende SQL-Statement

ATransaction : TSQLTransaction Die Transaktion in deren Kontext das SQL-Statement durchgeführt wird.

GetFieldNames

```
procedure GetFieldNames(const TableName : string; List : TStrings);
    virtual;
```

Ermittelt die Namen der Felder der Tabelle.

Methode von TxxxConnection>TSQLConnection

Parameter

TableName : string; Name der Tabelle deren Felder ermittelt werden sollen

List : TStrings; Die Liste der Felder

GetProcedureNames

```
procedure GetProcedureNames(List : TStrings); virtual;
```

Ermittelt die Namen der gespeicherten Prozeduren.

Methode von TxxxConnection>TSQLConnection

Parameter

List : TStrings; Die Liste der Prozeduren

GetTableNames

```
procedure GetTableNames(List : TStrings; SystemTables : Boolean
    = false); virtual;
```

Ermittelt die Tabellennamen oder die Systemtabellennamen (abhängig von SystemTables)

Methode von TxxxConnection>TSQLConnection

Parameter

List : TStrings; Die Liste der Tabellennamen

SystemTables : Boolean Ob Systentabellen (true) oder Usertabellen (false) ermittelt werden sollen.

Open

```
procedure Open;
```

Setzt ganz einfach die Eígenschaft active auf true.

Methode von TxxxConnection>TSQLConnection>TDatabase>TCustomConnection

StartTransaction

```
procedure StartTransaction; override;
```

Ruft die entsprechede Methode der Komponente Transaction auf.

Methode von TxxxConnection>TSQLConnection

CharSet

```
property CharSet : string read FCharSet write FCharSet;
```

Eigenschaft von TxxxConnection>TSQLConnection

Zugriff: Lesend und schreibend

Connected

```
property Connected: Boolean read FConnected write SetConnected;
```

Gibt an ob die Verbindung besteht oder nicht. True aktiviert die Verbindung.

Eigenschaft von TxxxConnection>TSQLConnection>TDatabase

Zugriff: Lesend und schreibend

DatabaseName

```
property DatabaseName: string read FDatabaseName write FDatabaseName;
```

Gibt den Namen der Datenbank an. Ist für die verschiedenen Datenbanken unterschiedlich. Siehe bei den entsprechenden Verbindungen.

Eigenschaft von TxxxConnection>TSQLConnection>TDatabase

Zugriff: Lesend und schreibend

HostName

```
property HostName : string Read FHostName Write FHostName;
```

Gibt den Namen des Hosts (Server) an auf welchen sich die Datenbank befindet. Bei lokalen Server ist das 'localhost' oder auch '127.0.0.1'.

Eigenschaft von TxxxConnection>TSQLConnection

Zugriff: Lesend und schreibend

KeepConnection

```
property KeepConnection;
```

ToDo

Eigenschaft von TxxxConnection>TSQLConnection>TDatabase

Zugriff: Lesend und schreibend

LoginPrompt

```
property LoginPrompt: Boolean read FLoginPrompt write FLoginPrompt;
```

Gibt an ob ein Login Prompt beim verbinden automatisch erzeugt werden soll.

Eigenschaft von TxxxConnection>TSQLConnection>TDatabase>TCustomConnect

Zugriff: Lesend und schreibend

Params

```
property Params : TStrings read FParams Write FParams;
```

Enthält spezielle Parameter für die Verbindung.

Eigenschaft von TxxxConnection>TSQLConnection>TDatabase

Zugriff: Lesend und schreibend

Password

Enthält das Passwort für die Verbindung. Nicht benötigt, wenn LoginPrompt true ist.

Eigenschaft von TxxxConnection>TSQLConnection>TDatabase

Zugriff: Lesend und schreibend

Role

```
Property Role : String read FRole write FRole;
```

ToDo

Eigenschaft von TxxxConnection>TSQLConnection

Zugriff: Lesend und schreibend

StreamedConnected

```
property Streamedconnected: Boolean read FStreamedConnected write
    FStreamedConnected;
```

ToDo

Eigenschaft von TxxxConnection>TSQLConnection>TDatabase>TCustomConnection

Zugriff: Lesend und schreibend

Transaction

```
property Transaction : TSQLTransaction read FTransaction write SetTransaction
```

Gibt die Transaktions Komponente an.

Eigenschaft von TxxxConnection>TSQLConnection

Zugriff: Lesend und schreibend

UserName

```
property UserName : string read FUserName write FUserName;
```

Enthält den Benutzernamen für die Verbindung beziehungsweise für den Zugriff auf die Datenbank.

Eigenschaft von TxxxConnection>TSQLConnection

Zugriff: Lesend und schreibend

Jede Datenbank hat ihre spezielle Verbindung (Connection).

3.1.3 TMySQL50Connection

Jede Datenbank hat ihre spezielle Verbindung (Connection). Dies hier ist die Verbindung zu MySQL 5.0

3.1.4 TMySQL41Connection

Jede Datenbank hat ihre spezielle Verbindung (Connection). Dies hier ist die Verbindung zu MySQL 4.1

3.1.5 TMySQL40Connection

Jede Datenbank hat ihre spezielle Verbindung (Connection). Dies hier ist die Verbindung zu MySQL 4.0

3.1.6 TOracleConnection

Jede Datenbank hat ihre spezielle Verbindung (Connection). Dies hier ist die Verbindung zu Oracle.

3.1.7 TPQConnection

Jede Datenbank hat ihre spezielle Verbindung (Connection). Dies hier ist die Verbindung zu PostGreSQL Datenbanken.

3.1.8 TODBCConnection

Jede Datenbank hat ihre spezielle Verbindung (Connection). Dies hier ist die Verbindung zu den ODBC Treibern.

3.1.9 TSQLTransaction

Als Transaktion wird eine feste Folge von Operationen, die eine Einheit bilden, bezeichnet. Transaktionen müssen die ACID-EIgenschaft garantieren. A) Atomität, das heißt untrennbar. Es wird entweder alles oder nichts durchgeführt. C) Konsistenz, Nach der Transaktion müssen die Daten konsistent sein. Daher auch an allen geänderten Stellen den gleichen Inhalt haben. I) Isolation, mehrere gelichzeitig laufende Transaktionen dürfen sich nicht gegenseitig beeinflussen. D) Dauerhaft, die Auswirkungen der Transaktion müssen im Datenbestand dauerhaft sein. Auch bei widrigen Umständen dürfen die Transaktionen nicht verloren gehen oder vergessen werden, zB. bei Rücksicherungen nach Absturz.

Der Ablauf einer Transaktion ist relativ einfach. Die Transaktion wird eröffnet, dann die Handlungen an der Datenbank gesetzt und die Transaktion entweder mit 'Rollback' wenn sie zurückgenommen werden soll oder mit 'Commit' wenn die Änderderungen dauerhaft übernommen werden sollten, abgeschlossen.

Man muß sich nur vor Augen halten, das das Datenbanksystem um die ACID Eigenschaften garantieren zu können, Aktionen wie Sperren, Duplizieren oder auch Warten durchführen muß. Deshalb soll eine Transaktion nur solange aufrecht erhalten werden wie es unbedingt nötig ist. Das heisst, auch, während einer Benutzereingabe oder sonstiger Wartezeit sollte keine Transaktion stattfinden. Besonders bei Mehrbenutzersystemen kann das bis zum Stillstand der Datenbank führen, wenn wegen vergessener Eingabe bei Arbeitscschluß eine Transaktion aktiv bleibt und deshalb eine Sperre auf einer Datenbank liegt.

Commit

```
procedure Commit; virtual;
```

Schliesst die Transaktion ab

Methode von TSQLTransaction

CommitRetaining

```
procedure CommitRetaining; virtual;
```

Führt ein Commit durch, lässt aber die Datenmenge offen. Wenn es die Datenbank nicht unterstützt, so wird es von der SQLdb Komponete simmuliert.

Methode von TSQLTransaction

EndTransaction

```
procedure EndTransaction; override;
```

Beendet die Transaktion, führt ein normalerweise ein Rollback durch.

Methode von TSQLTransaction

Rollback

```
procedure Rollback; virtual;
```

Rollt die Transaktion zurück, macht daher die Änderungen nicht aktiv.

Methode von TSQLTransaction

RollbackRetaining

```
procedure RollbackRetaining; virtual;
```

Führt ein Rollback durch, lässt aber die Datenmenge offen. Wenn es die Datenbank nicht unterstützt, so wird es von der SQLdb Komponete simmuliert.

Methode von TSQLTransaction

StartTransaction

```
procedure StartTransaction; override;
```

Startet die Transaktion

Methode von TSQLTransaction

Action

```
property Action : TCommitRollbackAction read FAction write FAction;
```

ToDo

Eigenschaft von TSQLTransaction

Zugriff: Lesend und schreibend

Database

```
Property DataBase : TDatabase Read FDatabase Write SetDatabase;
```

Gibt den Namen der Verbindung (=Datenbank) an.

Eigenschaft von TSQLTransaction>TDBTransaction

Zugriff: Lesend und schreibend

Params

```
property Params : TStrings read FParams Write FParams;
```

Enthält spezielle Parameter für die Verbindung.

Eigenschaft von TxxxConnection>TSQLConnection>TDatabase

Zugriff: Lesend und schreibend

3.1.10 TSQLQuery

Allgemeines

Der Name beschreibt nur unzureichend die Komponente. Es ist nicht nur ein Behälter für Abfragen (Query) sondern die Kapselung für die Datenmenge. Das heisst, über die Komponente

läuft eigenlich alles bezüglich Daten und Datenmenge. Sie wird sowohl für DDL[9], DML[10] und DCL[11] verwendet

Methoden von TSQLQuery

Im folgenden sind hier die wichtigsten Methoden beschrieben. Standardmethoden wie 'Create' oder 'Free' werden hier nicht beschrieben.

ApplyUpdates

```
procedure ApplyUpdates; virtual; overload;
```

```
procedure ApplyUpdates(MaxErrors: Integer); virtual; overload;
```

Überträgt die Änderungen von der Datenmenge in die Datenbank. 'ApplyUpdates' entspricht 'ApplyUpdates(0)'.

Methode von TSQLQuery>TBufDataset

CancelUpdates

```
procedure CancelUpdates; virtual;
```

Die Änderungen an der Datenmenge werden verworfen.

Methode von TSQLQuery>TBufDataset

Close

```
procedure Close;
```

Setzt die Eigenschaft 'Active' auf 'false'. Somit wird die Datenmenge geschlossen.

Methode von TSQLQuery

ExecSQL

```
procedure ExecSQL;
```

Führt die in der Eigenschaft SQL definierten SQL-Befehle aus, liefert aber keine Datenmenge zurück. Sonderfall 'Stored Procedure' kann unter umständen Datenmengen zurück liefern.

Methode von TSQLQuery

[9] Datendefinitionssprache siehe Kapitel 1.4.2 auf Seite 27
[10] Datenveränderungssprache siehe Kapitel 1.4.3 auf Seite 27
[11] Datenkontrollsprache siehe Kapitel 1.4.4 auf Seite 27

IsEmpty

```
function IsEmpty: Boolean;
```

Überprüft ob die Datenmenge leer ist. Gibt true zurück, wenn die Datenmenge leer ist. Zu beachten ist, das das Ergebnis immer false ist, wenn die Datenmenge im Zustand 'dsinsert' ist, sprich ein Datensatz eingefügt wird. Ausserdem ist die Prüfung nur dann sinnvoll wenn eine Abfrage aktiv ist.

Methode von TSQLQuery>TBufDataset>TDBDataset>TDataSet

Locate

```
function Locate(const keyfields: string; const keyvalues: Variant;
    options: TLocateOptions) : boolean; override;
```

Achtung, es ist derzeit nur möglich in einem Feld zu suchen, nicht in mehreren. Beim Suchen selbst sind keine Wildcards erlaubt ('*', '%',...)

Methode von TSQLQuery>TBufDataset

Parameter

keyfields: string Angabe des Suchfeldes

keyvalues: Variant Inhalt nach dem gesucht wird

options: TLocateOptions Scheibweise egal 'loCaseInsensitive', nur eine Teilmenge 'loPartialKey'

Open

```
procedure Open;
```

Führt die in der Eigenschaft SQL definierten SQL-Befehle aus und liefert die Datenmenge zurück. Intern wird die Eigenschaft 'Active' auf 'true' gesetzt.

Methode von TSQLQuery

Prepare

```
procedure Prepare;
```

Mittels Prepare wird die Komponente darauf vorbereitet, das es den Ablauf optimieren kann. Die Komponente geht davon aus, das das SQL-Statement endgültig ist, nur mehr die Parameter können sich ändern. Somit können die Statement verarbeitet werden und die Verbindungen vorbereitet. Die Komponente kann die Staements an die Server leiten und diese können die Befehle jetzt vorkompilieren und optimieren.

Methode von TSQLQuery

SetSchemaInfo

```
procedure SetSchemaInfo( SchemaType : TSchemaType; SchemaObject-
    Name, SchemaPattern : string);
```

Setzt das Schema, die aktuellen SQL Statements gehen dabei verloren und werden durch die neuen Informationen ersetzt. Zusätzlich wird der Datenbankzugriff auf nur lesen gesetzt.

Methode von TSQLQuery

Parameter

SchemaType : TSchemaType Kann folgende Werte annehmen: stNoSchema, stTables, stSysTables, stProcedures, stColumns, stProcedureParams, stIndexes, stPackages.

SchemaObjectName : string

SchemaPattern : string

Unprepare

```
procedure Unprepare;
```

Setzt die durch Prepare ausgelösten Vorverarbeitungen zurück. Sollte immer vor Änderungen an der Komponente aufgerufen werden, wenn man Prepare verwendet hat.

Methode von TSQLQuery

UpdateStatus

```
function UpdateStatus: TUpdateStatus; override;
```

Gibt den Status des Zwischenspeicher für geänderte Daten an. Kann folgende zustände annehmen : Keine Änderungen ist 'usUnmodified', bei Änderungen 'usModified', Einfügungen 'usInserted' und bei Löschungen 'usDeleted'.

Methode von TSQLQuery>TBufDataset

Eigenschaften von TSQLQuery

Hier beginnt die Beschreibung der wichtigsten Eigenschaften und Ereignisse. Wie schon bei den Methoden, werden hier nur die wichtigsten erläutert.

Active

```
property Active: Boolean read GetActive write SetActive default
    False;
```

Sagt aus ob die Komponente derzeit aktiv ist und eine Datenmenge zurückliefert. Ein setzen auf 'true' entspricht der Methode 'Open' und ein setzen auf 'false' entspricht der Methode 'Close'.

Eigenschaft von TSQLQuery

Zugriff: Lesend und schreibend

Defaultwert: false

Database

```
property Database;
```

Gibt die zuständige Databasekomponente (TxxxConnection) an.

Eigenschaft von TSQLQuery

Zugriff: Lesend und schreibend

DataSource

```
Property DataSource : TDatasource;
```

WIrd in einer Master-Detail Beziehung vom der Client-Query zur Synchronisierung mit der Master-Query verwendet verwendet.

Eigenschaft von TSQLQuery

Zugriff: Lesend und schreibend

Defaultwert: false

Filter

```
property Filter: string read FFilterText write SetFilterText;
```

Enthält den Filter für die Datenmenge, wenn definiert. Entspricht einer 'WHERE'-Klausel bei einem SQL Statement - siehe auch 'Serverfiltered'. Normalerweise ist eine Filterung am Server vorzuziehen, da die transportierte Datenmenge geringer ist. Wird erst aktiv durch aktivieren der Eigenschaft 'Filtered'.

Eigenschaft von TSQLQuery

Zugriff: Lesend und schreibend

Filtered

```
property Filtered: Boolean read FFiltered write SetFiltered default
    False;
```

Gibt an ob die Datenmenge gefiltert ist. Ein setzen der Eigenschaft bewirkt, das der Filter aktiv wird und die Datenmenge entsprechend eingeschränkt wird.

Eigenschaft von TSQLQuery

Zugriff: Lesend und schreibend

FilterOptions

```
property FilterOptions: TFilterOptions read FFilterOptions write
    SetFilterOptions;
```

Möglich sind 'foCaseInsensitive' ignoriert Groß-, Kleinschreibung und 'foNoPartialCompare' schaltet die Erkennung von Teilen beim Filtern aus. Wirkt nur auf die lokale Filterung und nicht auf die Serverfilter.

Eigenschaft von TSQLQuery>TBufDataset>TDBDataset>TDataSet

Zugriff: Lesend und schreibend

Params

```
property Params : TParams;
```

Enthält die Parameter für die SQLStatements

Eigenschaft von TSQLQuery

Zugriff: Lesend und schreibend

ParseSQL

```
property ParseSQL : Boolean;
```

Mit dieser Eigenschaft wird angezeigt ob die Komponente den SQL-Text auswerten soll. Soll sie den SQL-Text auswerten, so kann der Text für die Optimierung der Indexe etwas umgestellt werden. Weiters wird dann auch UsePrimaryKeyAsKey ausgewertet und zusätzlich, wenn nicht schon vorhanden, die Vorbereitungen für die UpdateSQl, DeleteSQL und InsertSQL durchgeführt. Es kann sein, das die Query in machen fällen schlecht ausgewertet wird, was sich durch dubiose SQLFehler zeigt. Dann kan man versuchen das ParseSQL und UsePrimaryKeyAsKey auszuschalten und alle Queries von Hand richtig zu setzen.

Eigenschaft von TSQLQuery

Zugriff: Lesend und schreibend

Prepared

```
property Prepared : boolean read IsPrepared;
```

Gibt den Status an, ob die Komponente Vorverarbeitung aktiviert hat.

Eigenschaft von TSQLQuery

Zugriff: Nur lesend

ReadOnly

```
property ReadOnly : Boolean;
```

Zeigt an ob auf die Datenmenge nur lesend zugegriffen werden kann.

Eigenschaft von TSQLQuery

Zugriff: Nur lesend

RecordCount

```
property RecordCount: Longint read GetRecordCount;
```

Ob die Eigenschaft verfügbar ist, hängt von der Connectionkomponente ab. Wenn nichts implementiert wird, so kommt bei einer Abfrage der Wert '-1' zurück. Ansonsten die Anzahl der Datensätze der Abfrage. Sinnvoller weise sollte eine Abfrage aktiv sein, ansonsten kann es zu Laufzeitfehlern kommen.

Eigenschaft von TSQLQuery>TBufDataset>TDBDataset>TDataSet

Zugriff: Nur lesend

ServerFilter

```
property ServerFilter: string read FServerFilterText write Set-
    ServerFilterText;
```

Enthält den Filter für die Datenmenge, wenn definiert. Intern wird der Filter in eine 'WHERE'-Klausel umgesetzt. Somit erfolgt die Filterung bereits am Server. Wird erst aktiv durch aktivieren der Eigenschaft 'ServerFiltered'.

Eigenschaft von TSQLQuery

Zugriff: Nur Lesend

ServerFiltered

```
property ServerFiltered: Boolean read FServerFiltered write Set-
    ServerFiltered default False;
```

Gibt an ob die Datenmenge am Server gefiltert ist. Ein setzen der Eigenschaft bewirkt, das der Filter am Server aktiv wird und die Datenmenge dort entsprechend eingeschränkt wird.

Eigenschaft von TSQLQuery

Zugriff: Lesend und schreibend

SQL, UpdateSQL, InsertSQL, DeleteSQL

```
property SQL : TStringlist;

property UpdateSQL : TStringlist;

property InsertSQL : TStringlist;
```

`property DeleteSQL : TStringlist;`

In SQL befindet sich das SQL Statement was ausgeführt werden soll. Ist es ein einfacheres Select-Statement so kann TSQLQuery auch automatisch die Statements für Änderungen (UpdateSQL), Einfügen (InsertSQL) und Löschen (DeleteSQL) ausfüllen. Ist das SQL Statement komplexer so geht diese automatik manchmal ins Leere, kann die Statements nicht richtig auswerten und verursacht unerklärliche Probleme.
UpdateSQL: Hier stehen die SQL Statements um Änderungen in der Datenmenge durchzuführen.
InsertSQL: Die Statements um Datensätze einzufügen.
DeleteSQL: Statements um Datensätze zu löschen.
Die SQL Statements interaktieren mit der Eigenschaft ReadOnly, UsePrimaryKeyAsKey, ParseSQL, IndexDefs und Params (Vielleicht auch einigen mehr). Wenn die Komponente nicht so reagiert wie erwartet, dann sollte man sich das SQL und die vorher genannten Eigenschaften näher ansehen.

Eigenschaft von TSQLQuery

Zugriff: Lesend und schreibend

Transaction

`property Transaction;`

Gibt die zuständige Transaktionskomponente an.

Eigenschaft von TSQLQuery

Zugriff: Lesend und schreibend

StatementType

`property StatementType : TStatementType read GetStatementType;`

Möglich sind 'stNone', 'stSelect', 'stInsert', 'stUpdate', 'stDelete', 'stDDL', 'stGetSegment', 'stPutSegment', 'stExecProcedure', 'stStartTrans', 'stCommit', 'stRollback' und 'stSelectForUpd'. Es wird hier der Komponente mitgeteilt, wie sie den Text in der Eingeschaft 'SQL' interpretieren soll.

Eigenschaft von TSQLQuery

Zugriff: Lesend und schreibend

UpdateMode

`property UpdateMode : TUpdateMode read FUpdateMode write SetUpdateMode;`

Setzt den Modus nach welcher die Datensätze geändert werden sollen. Möglich sind alle Datensätze 'upWhereAll', nur die geänderten Datensätze 'upWhereChanged' oder nur welche wo der Schlüssel bekannt ist 'upWhereKeyOnly'.

Eigenschaft von TSQLQuery

Zugriff: Lesend und schreibend

UsePrimaryKeyAsKey

```
property UsePrimaryKeyAsKey : boolean;
```

Wenn aktiv, so benutzt ParseSQL den Primärschlüssel immer als Schlüssel zu verwenden. Kann bei komplexeren Abfragen manchmal zu Problemen führen, da der Parser den SQL-Text nicht richtig auswerten kann. Siehe auch bei der Eigenschaft ParseSQL.

Eigenschaft von TSQLQuery

Zugriff: Lesend und schreibend

```
Version: $$ [12]
```

[12] Autor: Andreas Frieß
Lizenz: GFDL

3.2 utils.pas

3.2.1 Beschreibung

Einleitung Diese Unit wurde ürsprünglich von Christian Ulrich[11] geschrieben, weil diese Routinen in dieser oder ähnlicher Form auch nicht in der JCL[13][12] oder ähnlichen Sammlungen vorkommen.

Abhängigkeiten Es bestehen Abhängikeiten zu *Classes, SysUtils, Graphics, Forms, Process, Dialogs, Clipbrd, FileUtil, Translations* zusätzlich bei Windows noch zu *Registry* und *Windows*

3.2.2 Code

ClearDir

```
function ClearDir (Path: string): boolean;
```

Löscht ein Verzeichnis komplett. Die Funktion arbeitet rekursiv, daher löscht sie auch enthalten Verzeichnisse. Vorsicht ist geboten, da es keine weiteren Sicherheitsabfragen gibt, da ein Aufruf auf das Wurzelverzeichnis alles löscht.

Parameter

Path: string Der Pfad der zu löschen ist. Ein Abschliessender „/“ ist nötig.

Result

Rückgabe: boolean *true* wenn erfolgreich, *false* wenn ein Problem war.

DateTimeToHourString

```
function DateTimeToHourString(DateTime : TDateTime) : string;
```

Wandelt ein Datum in einen Stunden/Minuten String um wie zum Beispiel *05:30*.

Parameter

DateTime : TDateTime Das Datum das in ein Uhrzeit im Textformat gewandelt wird.

Result

Rückgabe: string Die in einen Uhrzeit String gewandelte Datum.

DateTimeToIndustrialTime

```
function DateTimeToIndustrialTime(dateTime : TDateTime) : string;
```

Wandelt ein Datum in einen industriellen Stunden/Minuten String um wie zum Beispiel *0530*.

Parameter

[13] JCL

DateTime : TDateTime Das Datum das in ein Uhrzeit im Textformat gewandelt wird.

Result

Rückgabe: string Die in einen Uhrzeit String gewandelte Datum.

DrawText

```
procedure DrawText(Canvas : TCanvas; Rect : TRect; Str : string;
    CenterV : Boolean = False; CenterH : Boolean = False);
```

Zeichnet den Text auf den Canvas.

Parameter

Canvas : TCanvas Der Canvas auf den gezeichnet wird

Rect : TRect Ein rechteckiger Bereich in den gezeichnet wird

Str : string Der String der gezeichnet werden soll

CenterV : Boolean Ob Vertikal zentriert wird. Standard ist nein

CenterH : Boolean Ob Horizontal zentriert wird. Standard ist nein

ExecProcessEx

```
function ExecProcessEx(CommandLine : string; CurDir : string = ")
    : string;
```

Führt ein externes Programm aus.

Parameter

CommandLine : string Die Kommandozeile, die an das externe Programm übergeben wird.

CurDir : string Das Verzeichnis das aktuellsein soll. Standard ist das aktuelle.

Result

Rückgabe: string Ist der von dem Programm auf de Kommandozeile ausgegeben Text, wenn Erfolgreich, ansonsten eine Fehlermeldung.

IsNumeric

```
FUNCTION IsNumeric(s: STRING): boolean;
```

Überprüft ob ein String in eine Zahl gewandelt werden kann.

Parameter

s: STRING Der String gewandelt werden soll.

Result

Rückgabe: boolean True wenn der Sting in eine Zahl gewandelt werden kann, false wenn er nicht gewandelt werden kann.

InstallExt

```
function InstallExt(Extension, ExtDescription, FileDescription,
    OpenWith, ParamString: string; IconIndex: Integer = 0): Boolean;
```

Ordnet einer Dateierweiterung ein Programm zu. Vergleich dem, das beim anklicken einer Textdatei immer ein Editor geöffnet wird.

Parameter

Extension: STRING Die Dateierweiterung die Zugeordnet werden soll

ExtDescription: STRING Ein kurze Beschreibung der Dateierweiterung

FileDescription: STRING Dateibeschreibung

OpenWith: STRING Wit welchen Programm geöffnet werden soll

ParamString: STRING Welche Parameter (Kommandozeile) übergeben werden

IconIndex: Integer Welches Icon (mit der Iconnummer, siehe *GetMainIconHandle* auf Seite 55) verwendet werden soll

Result

Rückgabe: boolean .

HTTPEncode

```
function HTTPEncode(const str : String) : string;
```

Beschreibung fehlt

Parameter

const str : String .

Result

Rückgabe: string .

ValidateFileName

```
function ValidateFileName(old : string) : string;
```

Beschreibung fehlt

Parameter

old : string .

Result

Rückgabe: string .

ValidateFileDir

```
function ValidateFileDir(old : string) : string;
```

Beschreibung fehlt

Parameter

old : string .

Result

Rückgabe: string .

ValidateDate

```
function ValidateDate(D : string) : string;
```

Beschreibung fehlt

Parameter

D : string .

Result

Rückgabe: string .

GetTempPath

```
function GetTempPath : string;
```

Beschreibung fehlt

Result

Rückgabe: string .

GetConfigDir

```
function GetConfigDir(app : string) : string;
```

Beschreibung fehlt

Parameter

app : string .

Result

Rückgabe: string .

GetGlobalConfigDir

```
function GetGlobalConfigDir(app : string) : string;
```

Beschreibung fehlt

Parameter

app : string .

Result

Rückgabe: string .

SizeToText

```
function SizeToText(size : Longint) : string;
```

Beschreibung fehlt

Parameter

size : Longint .

Result

Rückgabe: string .

GetMainIconHandle

```
function GetMainIconHandle : Cardinal;
```

Gibt das Handle auf das Haupt Icon der Anwendung zurück. Das ist das Icon, das in der Taskleiste beziehungsweise am linken oberen Fensterrand befindet.

Result

Rückgabe: Cardinal Das Handle auf das Haupt Icon der Anwendung.

CanWriteToProgramDir

```
function CanWriteToProgramDir : Boolean;
```

Beschreibung fehlt

Result

Rückgabe: boolean .

OpenBrowser

```
function OpenBrowser(Site : string) : Boolean;
```

Beschreibung fehlt

Parameter

Site : string .

Result

Rückgabe: boolean .

HexToBin

```
function HexToBin(h: STRING): dword;
```

Beschreibung fehlt

Parameter

h: STRING .

Result

Rückgabe: dword .

LoadLanguage

```
procedure LoadLanguage(lang : string);
```

Beschreibung fehlt

Parameter

lang : string .

RoundTo

```
function RoundTo(const AValue : extended ; const ADigit : TRoundToRange)
    : extended ;
```

Beschreibung fehlt

Parameter

AValue : extended .

ADigit : TRoundToRange .

Result

Rückgabe: extended .

TimeTotext

```
function TimeTotext(Seconds : Integer) : string;
```

Beschreibung fehlt

Parameter

Seconds : Integer .

Result

Rückgabe: string .

ExecProcess

```
procedure ExecProcess(CommandLine : string;CurDir : string = ';Waitfor
    : Boolean = True);
```

Beschreibung fehlt

Parameter

CommandLine : string .

CurDir : string .

Waitfor : Boolean .

ExecVisualProcess

```
procedure ExecVisualProcess(CommandLine : string;CurDir : string
    = '';Waitfor : Boolean = True);
```

Beschreibung fehlt

Parameter

CommandLine : string .

CurDir : string .

Waitfor : Boolean .

GetProcessforExtension

```
function GetProcessforExtension(InfoTyp : TProcessinfoTyp;Extension
    : string) : string;
```

Beschreibung fehlt

Parameter

InfoTyp : TProcessinfoTyp .

Extension : string .

Result

Rückgabe: string .

GetMimeTypeforExtension

```
function GetMimeTypeforExtension(Extension : string) : string;
```

Beschreibung fehlt

Parameter

Extension : string .

Result

Rückgabe: string .

GetSystemLang

```
function GetSystemLang : string;
```

Beschreibung fehlt

Result

Rückgabe: string .

RPos

```
function RPos(const Substr: string; const S: string): Integer;
```

Beschreibung fehlt

Parameter

Substr: string .

S: string .

Result

Rückgabe: Integer .

StripHTML

```
function StripHTML(S: string): string;
```

Beschreibung fehlt

Parameter

S: string .

Result

Rückgabe: string .

StrTimeToValue

```
FUNCTION StrTimeToValue(val : string) : LongInt;
```

Beschreibung fehlt

Parameter

val : string .

Result

Rückgabe: LongInt .

SystemUserName

```
function SystemUserName : string;
```

Beschreibung fehlt

Result

Rückgabe: string .

Version: $LastChangedRevision: $ [14]

[14] Autor: Andreas Frieß
Lizenz: GFDL

3.3 uFileMisc.pas

3.3.1 Beschreibung

Einleitung Diese Unit wurde von Erhard Kieling[14] geschrieben und dient zur Unterstützung bei Suchvorgängen. Beispiele zur Verwendung befinden sich im LazSnippets Code Paket auf Sourceforge.

Abhängigkeiten Es bestehen Abhängikeiten zu *Classes, SysUtils, StdCtrls*.

3.3.2 Typendefinitionen

```
TSampleSearchResults = procedure (ADir, AFileName: string) of object;
```

Definition der Callbackfunktion

Parameter

ADir: string Das Verzeichnis in dem in dem die Datei gefunden wurde.

AFileName: string Der Dateiname der gefunden wurde.

3.3.3 Code

SearchFile Variante 1

```
function SearchFile(AInitDir, AFileName : string; ShowContent: TSampleSearchR
    string; overload;
```

Sucht in einem Verzeichnis nach einer Datei.

Parameter

AInitDir: string Das Verzeichnis in dem gesucht werden soll.

AFileName: string Der Dateiname der gesucht wird, Wildcards sind erlaubt.

ShowContent: TSampleSearchResults Callbackfunktion die für jedes Ergebnis aufgerufen wird. Sie muß nicht zwingend angegeben werden. Wird sie nicht verwendet so ist *nil* anzugeben.

Result

Rückgabe: string Dateiname mit Pfadangaben.

SearchFile Variante 2

```
function SearchFile(AInitDir, ASubDir, AFileName : string; ShowContent:
    TSampleSearchResults): string; overload;
```

Parameter

AInitDir: string Das Verzeichnis in dem gesucht werden soll.

ASubDir: string Das Unterverzeichnis in dem auch gesucht werden soll.

AFileName: string Der Dateiname der gesucht wird, Wildcards sind erlaubt.

ShowContent: TSampleSearchResults Callbackfunktion die für jedes Ergebnis aufgerufen wird. Sie muß nicht zwingend angegeben werden. Wird sie nicht verwendet so ist *nil* anzugeben.

Result

Rückgabe: string .

3.3.4 Beispiel

```
procedure TForm1.SpeedButton1Click(Sender: TObject);
begin
  if SelectDirectoryDialog1.Execute then begin
    edStartDir.Text:= SelectDirectoryDialog1.FileName;
  end; // of if SelectDirectoryDialog1.Execute then begin
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
  edStartDir.Text:= PathDelim;
end;

procedure TForm1.SampleResults(ADirName, AFileName: string);
begin
  if (ADirName <> '') then begin
    lbDirBox.Items.Add(ADirName);
    Self.Update;
  end;
  if (AFileName <> '') then begin
    lbFileBox.Items.Add(AFileName);
    Self.Update;
  end;
end; // of TForm1.SampleResults

procedure TForm1.Button2Click(Sender: TObject);
var
  s: string;
begin
  lbDirBox.Clear;
  lbFileBox.Clear;
  s:= SearchFile(edStartDir.Text,
                 '',
                 edFileName.Text,
```

```
                  @SampleResults);
  if (s <> '') then edFileName.Text:= s;
end;
```

Version: $LastChangedRevision: $ [15]

[15] Autor: Andreas Frieß
Lizenz: GFDL

4 Beispiele

4.1 DatenbankenMySQL5x

4.1.1 Demodatenbank MySQL

Installieren von MySQL 5.x

Am einfachsten ist es im Internet auf die MySQL-Seite[1] zu gehen und dort unter „Community“[2] den Punkt „Downloads“ und „MySQL Community Server“ auszusuchen. Dort kann man sich dann den entsprechenden Server für sein Betriebssystem aussuchen und herunterladen. Die Dateigröße kann schon einen Breitband Internetzugang verlangen, denn je nach Version sind bis zu knappen 100 MB Download-Volumen gefragt.

Weiters ist es zu empfehlen, das man sich unter „Downloads, GUI Tools“ auch noch die Programme „MySQL Administrator“ und „MySQL Query Browser“ herunterlädt. Mit Hilfe der beiden Tools kann man später dann den Server komfortabel administrieren und auch die Datenbanken verwalten und Scripts laufen lassen. Die Tools sind nicht zwingend erforderlich, erleichtern aber gerade am Anfang das Leben.

Eine Alternative dazu ist sicherlich auch das arbeiten mit „PHPMySQLAdmin“. Das Tool ist sehr ausgereift und auch entsprechend bekannt. Ein Nachteil dabei ist, das es einen Webserver mit PHP voraussetzt.

Zur Installation geht man entsprechend den Erfordernissen seines Betriebssystems vor. Es würde den Rahmen dieser Beschreibung sprengen, für jedes Betriebssystem mit seinen Eigenheiten eine entsprechende Anleitung zu erstellen.

Wichtig ist nur, das man bei der Installation und Konfiguration nicht vergisst, ein gutes Passwort für den „root“ („Administrationsuser“) zu wählen. Denn später vergisst man gerne den laufenden MySQL-Server und wenn der Rechner dann doch im Internet auftaucht, so hat man ein ganz schönes Sicherheitsleck, das einem gar nicht bewusst ist. Dasselbe gilt natürlich in abgemilderter Form auch für normale Benutzer, die etwas mehr Rechte als lesen in der Datenbank haben. Eine gute Möglichkeit ist, ein launiges Sprichwort zu nehmen und die Anfangsbuchstaben aneinander zu reihen. Beispiel: „Wir sind ja nicht blöd Mann und haben 99 Luftballone gekauft“. Daraus kann sich ein Paßwort wie folgt ergeben „WsjnbMuh99Lg“. Das lässt sich noch so halbwegs merken und besteht aus Groß und Kleinbuchstaben, beinhaltet Zahlen und ist außerdem noch zwölf Zeichen lang.

Aber genug dem Ausflug in die Installation und Paßwortauswahl, beginnen wir nun mit der Erstellung unserer Datenbank, die sich dann später durch alle Beispiele ziehen wird.

[1] http://www.mysql.de/
[2] http://dev.mysql.com/

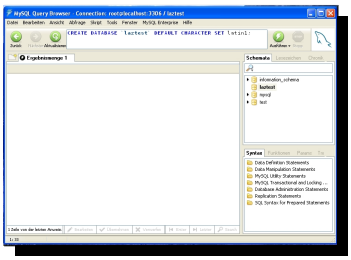
Erstellung der DEMO Datenbank

Das Erste, das wir am neuen oder geleerten MySQL Datenbankserver erstellen, ist eine neue Datenbank für unsere Zwecke. Dazu öffnen wir den „MySQL Query Browser“.

Es öffnet sich als ersters das Verbindungsfenster. Ich werde jetzt die Felder von oben nach unten besprechen, denn es könnte ja sein, das sie eine andere Sprachversion verwenden.

Unter „Gesp. Verbindungen“ wird ein beliebiger Namen eingetragen, unter der später die Einstellungen wieder abgerufen werden können. Bei „Server Host“ und „Port“ geben wir, bei einer lokalen Installation, „localhost“ und „3306“ ein. Wobei die Portnummer „3306“ der Standardport von MySQL ist. Als nächstes kommen jetzt der „Nutzername“ und das „Passwort“ dran, dort geben wir in diesem Fall den Benutzer „root“ mit dem bei der Installation vergebenen Paßwort ein. Später sollte man sich nur in zwingend notwendigen Fällen mit dem Administratorpaßwort verbinden. Das Feld „Standardschema“ lassen wir bei diesem ersten Einstieg einmal leer.

Anschliessend gehen wir mit „Ok“ im Dialog weiter, das nun folgende Hinweisfenster zum Thema fehlendes Standardschema übergehen wir mit dem Button „Ignorieren“. Ist bis jetzt alles gut verlaufen und der MySQL Datenbakserver aktiv, so öffnet sich jetzt endgültig der „MySQL Query Browser“.



Als ersters erzeugen wir mittels ***CREATE USER lazarus@localhost IDENTIFIED BY 'WsjnbMuh99Lg';*** einen neuen Benutzer, der noch dazu ein halbwegs sicheres Paßwort hat. Zusätzlich geben wir diesen Benutzer alle Recht an dieser Datenbank mit folgenden Befehl ***GRANT all ON *.* TO lazarus@localhost;***. Somit hat der Benutzer alle Rechte außer dem Recht, selbst Rechte zu vergeben.

Im Menüpunkt „Datei“ wählen wir die Funktion „Verbindung umschalten“ an und wechseln die Verbindung auf den User „lazarus@localhost“. Alternativ kann man den „MySQL Query Browser“ schliessen und als Benutzer „lazarus“ neu anmelden. Damit wir nicht immer die Informationen neu eingeben müssen, kann man den Button rechts neben den „Gesp. Verbindungen“ benutzen und sich dort ein entsprechndes Profil anlegen. Somit braucht man später nur sein Profil in der Top-Down Box anwählen, das Paßwort eingeben und schon ist man drinnen.

Anschließend erzeugen wir jetzt als Benutzer „lazarus@localhost“ mittels dem Kommando ***CREATE DATABASE 'laztest' DEFAULT CHARACTER SET latin1;*** eine neue Datenbank, mit dem Namen „laztest“. Diese Zeile geben wir oben im „MySQL Query Browser“ ein und drücken dann auf den grünen Button daneben. Anschließend können wir mit der Maus auf „Schemata“ klicken und mit der Taste „F5“ ein neuerliches einlesen der Übersicht der Datenbanken (= Schemata) erreichen. Jetzt sollte dort unsere neue Datenbank vorhanden sein.

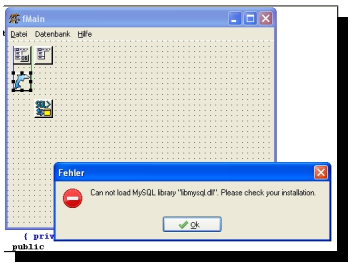
Somit haben wir die Datenbank erstellt und auch einen Beutzer mit hohen Rechten darin erzeugt. Für den produktiven Betrieb würde sich jetzt noch anbieten, einen Benutzer zu erzeugen,

der entsprechend den Erfordernissen weniger Rechte hat. Aber nachdem das hier ja ein Tutorium ist, belassen wir es einmal dabei.

Windows FAQ

Einleitung Es hat sich auch bei mir gezeigt, das es manchmal nicht so einfach ist, MySQL unter Lazarus zu verbinden. Hier ein paar Hilfen.

Cannot load MySQL library „libmysql.dll“ Dieser Fehler kann mehrer Ursachen haben. Die einfachste ist, dass auf dem Rechner ganz einfach die Treiber für MySQL nicht installiert sind.



Die Abhilfe ist die Datei in ein Verzeichnis, dass sich im Suchpfad befindet zu kopieren. Genauso ist es möglich, die Bibliothek ins selbe Verzeichnis wie die ausführbare Datei zu kopieren, der Weg bietet sich an, wenn es an den Rechten am Systemverzeichnis fehlt.

Wenn das ganze, wie am Bild, innerhalb von Lazarus passiert, so kann es einmal am Lazarus liegen. Besonders wenn man eine Version aus dem SVN selbst kompiliert, kann es vorkommen, das es hier Probleme gibt. Die Abhilfe ist nur, eine bessere oder stabile Version von Lazarus zu verwenden. Genauso kann es sein, dass wie oben bereits besprochen, die Biblothek fehlt. Die Abhilfe ist wieder die Biblothek in ein Verzeichnis, dass im Suchpfad liegt zu kopieren. Geht das nicht, so muß die Datei einmal in des Verzeichnis kopiert werden, wo sich die „Lazarus.exe“ befindet und einmal in das Verzeichnis, wo die ausführbare Datei des Projektes liegt. Denn innerhalb der IDE benötigt Lazarus die Datei in seinem Pfad, wird die Datei aber ausgeführt, so benötigt sie das aktuelle, kompilerte und gestartete Programm.

`Version: $LastChangedRevision: 38 $` [3]

4.1.2 Projekt MySQLSimple

Einleitung In diesem Projekt wird nur eine einfache Verbindung zur Datenbank aufgebaut und die grundlegenden Elemente die dafür notwendig sind erklärt. Es werden hierbei nur Elemente verwendet die bei einer Standardinstallation von Lazarus dabei sind.

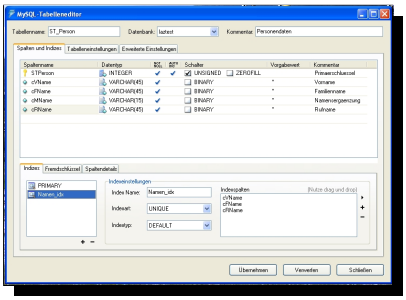
Datenbank In der Datenbank muß für dieses Beispiel folgende Anweisung ausgeführt werden oder das ganze mittels des „MySQL Query Browserërstellt werden.

```
CREATE TABLE `laztest`.`ST_Person` (
  `STPerson` INTEGER UNSIGNED NOT NULL AUTO_INCREMENT \
    COMMENT 'Primaerschluessel',
  `cVName` VARCHAR(45) NOT NULL DEFAULT '' COMMENT 'Vorname',
  `cFName` VARCHAR(45) NOT NULL DEFAULT '' COMMENT 'Familienname',
```

[3] Autor: Andreas Frieß
Lizenz: GFDL

```
  `cMName` VARCHAR(15) NOT NULL DEFAULT '' COMMENT 'Namensergaenzung',
  `cRName` VARCHAR(45) NOT NULL DEFAULT '' COMMENT 'Rufname',
  PRIMARY KEY (`STPerson`),
  UNIQUE INDEX `Namen_idx`(`cVName`, `cFName`, `cRName`)
)
ENGINE = InnoDB
CHARACTER SET latin1 COLLATE latin1_german1_ci
COMMENT = 'Personendaten';
```

Wir erstellen hier einmal die für das Beispiel notwendige Tabelle für Personendaten.



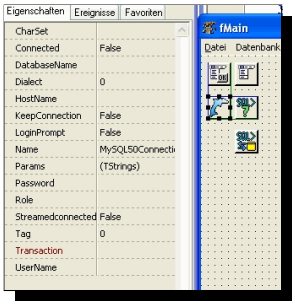
Warum habe ich den Namen „ST_Person“ gewählt. Ganz einfach, ich rechne Tabellen mit diesem Inhalt zu den so genannten Stammdatentabellen. Das stimmt, außer wir haben hier eine reine Personenverwaltung. Nachdem das Beispiel aber erweitert werden soll, werden auch noch andere Tabellen folgen.

Wir verwenden hier einmal die Spalte „STPerson“ für den nicht Null beinhaltenden (NOT NULL), automatisch hochzählenden (AUTO_INCREMENT), eindeutigen Primärschlüssel (PRIMARY KEY ('STPerson')). In der Spalten „cVName“ und „cFName“ kommen der Vorname und der Familienname (=Nachname) hinein, zusätzlich gibt es noch das Feld für einen eventuelle Namensergänzung. Weiters beinhaltet die Spalte „cRName“ noch den Rufnamen der Person.

Fast ganz unten in der definition der Tabelle befindet sich ein Hinweis auf einen weiteren Index. Dieser eindeutige Index (UNIQUE INDEX) soll verhindern, das gleiche Personen mehrmals in der Tabelle angelegt werden können. Warum wird dann aber nicht nur die Felder des Vornamens und des Familienanmens herangezogen ? Na, ja, was ist wenn wir mehrere „Max Mustermann“ haben ! Im richtigen Leben hat man dann ja selbst noch hilfen, das zu unterscheiden. Dazu dient der Rufname (= Spitzname). Denn dann kannman den „Max aus D“ und den „Max aus A“ auch noch eintragen und die EInträge sind jetzt trotz der Namensgleichheit wirklich auseinander zu halten.

Benutzerschnittstelle Die Komponeneten kommen von der Palettenseite „SQLdb“, „DataAccess“ und „DataControls“.

Fangen wir einmal mit der „SQLdb“ an. Um auf eine Datenbank zugreifen zu können, benötigen wir einmal eine Verbindungskomponente, das ist in unseren Fall eine der zur Datenbank passende Connection (= Verbindung). Da wir einen MySQL Server 5.x am laufen haben, so nehmen wir die passende „MySQL50Connection“ Komponente und bringen sie auf das Projekt.



Weiters fügen wir noch eine „TSQLTransaction“ Komponente unserer Form hinzu.

Jetzt der Reihe nach, angefangen bei der „TMySQL50Connection“ Komponente, müssen wir Einstellungen im Objektinspektor treffen.

Bei der „TMySQL50Connection“ Komponente sind es folgene Einstellungen. Der **DatabaseName** wird auf den Namen unserer Datenbank *laztest* gesetzt. Der **HostName** ist bei einer kokalen installation immer *localhost*. Für das **Paßwort** nehmen wir das des Benutzers, der sich mit der Datenbank verbindet. Bei **Transaction** müssen wir nur in das im Objektinspektor klicken und eine Auswahl erscheint. Wir nehmen hier unsere *SQLTransaction1*. Als letztes geben wir den **UserNamen** ein, das ist ganz einfach der Benutzer den wir in der Datenbank erstellt haben, also *lazarus@localhost*.

Die nächste Komponenete ist die „TSQLTransaction“. Dort müssen wir aktuell nichts einstellen, da sich die Eigenschaft **Database** im Objektinspektor bereits selbst auf *MySQL50Connection1* geändert hat.

Den ersten Test kann man jetzt bereits durchführen. Durch anklicken der Eigenschaft **Connected** von **MySQL50Connection1** wechselt diese von *False* auf *True*. Wenn alles gut gelaufen ist, so läuft die Verbindung ohne Probleme ab. Wenn nicht, so muß man die Einstellungen nochmals überprüfen. Anschliessen stellen wir die Eigenschaft wieder auf *False* zurück.

Nachdem die Verbindung zur Datenbank möglich ist, so müssen wir jetzt die Komponeneten für die Abfrage der Datenbank und die Anzeige der Daten noch hinzufügen. Es handelt sich hier um die Komponeneten „TSQLQuery“ von Tab „sqldb“, „TDataSource“ vom Tab „Data Access“ und „TDBGrid“ vom Tab „Data Controls“.

Fangen wir mit der Komponente „SQLQuery1“ an. Sie ist zuständig für die richtige Abfrage am Datenbankserver. Wir stellen die Eigenschaft **DataBase** auf unsere *MySQL50Connection1* ein. Weiters müssen wir in die Eigenschaft **SQL** folgendes einfügen

```
SELECT `STPerson`, `cVName`,
       `cFName`, `cMName`, `cRName`
  FROM st_person;
```

und den Eigenschaftseditor wieder schließen. Die Eigenschaft hat sich **Transaction** im Objektinspektor bereits selbst auf *SQLTransaction1* geändert. In der Komponenete „Datasource1“ müssen wir nur **DataSet** per Mausklick auf *SQLQuery1* einstellen. Als letztes konfigurieren wir das „TDBGrid“. Wir ändern mit der Maus die Eigenschaft **Align** auf *alClient* und die Eigenschaft **DataSource** auf *Datasource1*. Damit erfasst das Grid das komplette Formular.

Prinzipiell ist unser Programm jetzt einmal optisch fertig. Wenn wir es jetzt kompilieren und laufen lassen, sehen wir das Formular, aber keine Daten und ändern können wir auch nichts. Es ist klar, wir haben die Komponenten konfiguriert, aber etwas Code wird auch benötigt um den Komponenten zu sagen, was wir wirklich wollen.

Dazu verwenden wir eine „TActionList“ und ein „TMainmenu“ aus dem Tab „Standard“. In der **ActionList1** erzeugen wir ganz einfach die Ereignisse *actCloseDB*, *actOpenDB* und *actDataSetRefresh*. Ausserdem Erzeugen wir die Ereignisproceduren für das Form erzeugen un Form schliessen. Im MainMenu Komponente erzeugen dann ein kleines Menü.

```
procedure TfMain.actOpenDBExecute(Sender: TObject);
begin
  if not MySQL50Connection1.Connected then
  begin
    MySQL50Connection1.Connected := true;
  end;
  if not SQLQuery1.Active then SQLQuery1.Active := true;
end;
```

Beim Eintreffen des Ereignisses für das Öffnen der Datenbank, wird als erstes Abgefragt ob die Verbindung bereits besteht. Ist die Verbindung vorhanden, so wird die Abfrage selbst geöffnet, falls sie noch nicht offen war.

```
procedure TfMain.actCloseDBExecute(Sender: TObject);
begin
  if MySQL50Connection1.Connected then
  begin
    if SQLQuery1.Active then
    begin
      SQLQuery1.ApplyUpdates;
      SQLQuery1.Active := false;
    end;
    MySQL50Connection1.Connected := false;
  end;
end;
```

Wenn die Datenbank wieder geschlossen werden soll, so sind zuerst die noch nicht gespeicherten Daten zurückzuschreiben, dies geschieht mit dem Kommando *ApplyUpdates*. Ohne diese Kommando bleiben die Änderungen nur lokal und werden mit der nächsten Abfrage wieder überschrieben. Anschliessend wird die Abfrage deaktiviert, dannach die Verbindung.

```
procedure TfMain.actDataSetRefreshExecute(Sender: TObject);
begin
  if not SQLQuery1.Active then exit;
  SQLQuery1.ApplyUpdates;
  SQLQuery1.Refresh;
end;
```

Bei einer einfachen Auffrischung der Daten, arbeiten wir hier vorsichtshalber die Änderungen ein, bevor wir die eigentliche Auffrischung durchführen. Falls die Abfrage nicht aktiv war, brauchen wir klarerweise gar nichts zu machen.

```
procedure TfMain.FormDestroy(Sender: TObject);
begin
  actCloseDBExecute(Sender);
end;
```

Wichtig ist, das beim Schließen des Formular, die Datenbankkomponenten wieder ordnungsgemäß von der Datenbank getrennt werden, ansonsten hat man Fehlermeldungen und auch blockierte Datenverbindungen. Generell sollte danch getrachtet werden, die Verbindungen, egal was geschieht, in konsistenten Zustand zu hinterlassen.

SVN Den Quelltext des Beispiel kann man sich auch mit folgenden Kommando aus dem SVN ind das aktuelle Verzeichnis holen.

```
svn co https://lazsnippets.svn.sourceforge.net/svnroot/lazsnippets/\
  trunk/datenbank/MySQL/MySQLSimple .
```

Die Zeile ist in einem zu schreiben und wurde nur aus Formatierungsgründen hier beim Rückstrich (Backslash) umgebrochen. Der Punkt am Ende der Zeile ist notwendig, da es das Kennzeichen für das aktuelle Verzeichnis ist. Weiter ist auf die Schreibweise zu achten, das hier die Server zwischen Großbuchstaben und Kleinbuchstaben unterscheiden. Die Datenbank muß man aber trotzdem selbst am MySQL Server erstellen. Die dazu benötigten Informationen befinden sich im Text von „Datenbank“ weiter oben.

Version Derzeit ist das Beispiel mit der Version 0.9.23 beta von Lazarus entwickelt.

Betriebssystem	getestet
Linux Suse	nein
WinXP	ja

Abbildung 4.1: Versionsübersicht

`Version: $LastChangedRevision: 38 $` [4]

[4] Autor: Andreas Frieß
Lizenz: GFDL

4.1.3 Projekt MySQLTestData

Einleitung Es soll zeigen, wie man relativ einfach Testdaten in größerer Zahl in die Tabelle ST_Person unserer Datenbank einfügt. Zugleich kann man es als Basis für Versuche mit SELECT Statements verwenden. In der Tabelle werden Lösch-, Einfüge- und Auswahloperationen durchgeführt.

Datenbank Es wird die Datenbank und Tabelle vom Beispiel **MySQLSimple** verwendet. Weitere Informationen können dort gefunden werden.

Benutzerschnittstelle Auf der Oberfläche befinden sich links drei Buttons.

Abbildung 4.2: Projekt TestData GUI

Mit dem obersten Button wird die Tabelle komplett geleert, diese wird durch eine *Delete* Anweisung im Quelltext durchgeführt. Der mittlere Button und das Eingabefeld darüber gehören zusammen und fügen neue Datensätze in der Tabelle ein. Die Anzahl der Datensätze wird durch den Wert im Eingabefeld vorgegeben. Der unterste Button und das Tab-Sheet rechts daneben diesen zum Abfragen der Daten. Auf der ersten Seite des Tab-Sheets kann man das Abfrage Statement eingebn, die Anzeige erfolgt auf der zweiten Seite des Tab-Sheet.

Und jetzt die Funktionen mit den dahinterliegenden Code im Detail.

Oberer Button Zuerst wird geprüft ob die Verbindung (Connection) zur Datenbank besteht, wenn nicht so wird die Verbindung hergestellt. Die Verbindungseinstelllungen wurden über den Objektinspektor schon beim Design getroffen, da es sich hier um Beispiele handelt.

```
if not MySQL50Connection1.Connected then
  MySQL50Connection1.Connected := true;
aSqlQuery := TSQLQuery.Create(self);
try
  aSqlQuery.SQL.Clear;
  aSqlQuery.DataBase := MySQL50Connection1;
  aSQLQuery.Transaction := SQLTransaction1;
  aSqlQuery.SQL.Add('DELETE FROM st_person;');
  aSqlQuery.ExecSQL;
finally
  aSqlQuery.free;
end;
```

Anschliessend wird zur Laufzeit die TSQLQuery Komponente erzeugt. Wichtig ist, daß das Objekt erzeugt wird `(aSqlQuery := TSQLQuery.Create(self);)`. Durch *self* wird das Objekt dem Formular zugeordnet, so das das Formular beim Beenden auch das Objekt zerstören kann, falls es bis dahin noch nicht zerstört wurde.

Im nächsten Schritt wird ein eventuell vorhandenes gespeichertes SQL-Statement sicherheitshalber gelöscht. Weiters die Verbindung und die Transaktion zugewiesen. Dies geschieht genauso als würde man es über den Objektinspektor machen. Dann wird das neue SQL-Statement hinzugefügt. Da es ohne 'WHERE' EInschränkung verwendet wird, löscht es somit **alle** Daten aus der Tabelle.

Zum Abschluß wird das verwendete SQL-Query Objekt wieder sauber entfernt. Damit dies auch geschieht wenn es einen Fehler gegeben hat, sind die Teile ab der Erstellung des Objektes durch ein try..finally Statement geklammert. Damit wird erreicht, das das Objekt auch nach einem Fehler richtig gelöscht wird.

Mittlerer Button Beim betätigen des Buttons wird als erstes die Eingabe des Editfeldes überprüft und nur wenn der Wert in eine Zahl umgewndelt werden kann, wird fortgefahren.

```
try
  iAnzahl := StrToInt(edAnzahl.Text);
except
  showmessage('Nur Zahlen erlaubt');
  exit;
end;
```

Mittels des Behfels *randomize* wird der Zufallszahlengenerator initialisiert, dann die Verbindung zur Datenbank nötigenfalls hergestellt. Anschliessend eine TSQLQuery zur Laufzeit erzeugt und mit den nötigen Informationen versorgt.

```
randomize();
if not MySQL50Connection1.Connected then
  MySQL50Connection1.Connected := true;

aSqlQuery := TSQLQuery.Create(self);
try
  aSqlQuery.SQL.Clear;
  aSqlQuery.DataBase := MySQL50Connection1;
  aSQLQuery.Transaction := SQLTransaction1;
```

Weil wir später in einer Schleife Daten in die Datenbank einfügen, so erzeugen wir hier jetzt die nötigen Parameter.

HINWEIS: Das Arbeiten mit Parametern ist wesentlich besser, als das Statement mittels Stringverwaltung zusammen zu setzen. Man vermeidet damit Probleme mit speziellen Zeichen, wie Hochkomma und der SQL-Server hat die Möglichkeit das Statement vor zu kompilieren und zu optimieren.

Anschliessend wird das SQL-Statement in die Query eingefügt. Mit den Doppelpunkt zeigt man, das es sich hier um einen Parameter handelt.

```
  aSqlQuery.Params.CreateParam(ftInteger,'vname',ptInput);
  aSqlQuery.Params.CreateParam(ftInteger,'fname',ptInput);
```

```
  aSqlQuery.Params.CreateParam(ftInteger,'mname',ptInput);
  aSqlQuery.Params.CreateParam(ftInteger,'rname',ptInput);
  aSqlQuery.SQL.Add('INSERT st_person (cVName,cFName,cMName, cRName)');
  aSqlQuery.SQL.Add(' VALUES (:vname,:fname,:mname, :rname);');
```

Hier beginnt die Schleife. ALs erstes werden dir Stringvariablen erzeugt, anschliessend die Strings den Parameteren übergeben.

```
  for i := 1 to iANzahl do
  begin
    iZufall := Random(MaxVorNamen-MinVorNamen) + MinVorNamen;
    cVName := VorNamen[iZufall];
    if cVName = '' then continue;

    iZufall := Random(MaxFamilienNamen-MinFamilienNamen) +
                 MinFamilienNamen;
    cFName := FamilienNamen[iZufall];
    if cFName = '' then continue;

    cMName := leftStr(cVName,2)  +LeftStr(cFName,2);
    cRName := cVName + IntToStr(Random(10)) + cFName;

    aSqlQuery.Params.ParamByName('vname').AsString := cVName;
    aSqlQuery.Params.ParamByName('fname').AsString := cFName;
    aSqlQuery.Params.ParamByName('mname').AsString := cMName;
    aSqlQuery.Params.ParamByName('rname').AsString := cRName;
```

Jetzt wird die Query mit den aktuellen Parametern durchgeführt. Der Befehl lautet *ExecSQL*, dieser erwartet keine Ergebnismenge zurück. Das ist auch der Grund warum hier kein *Open* oder *Active=true* verwendet wurde, denn dieses erwarted das eine Ergebnismenge zurück geliefert wird. Anschliessend wird solange in der Schleife verblieben, bis alles abgearbeitet wurde. Wenn beim Ausführen ein Fehler auftritt, so wird in der Statuszeile eine Fehlermeldung angezeigt und mit dem nächsten durchlauf weitergemacht.

```
    try
      aSqlQuery.ExecSQL;
    except
      Statusbar1.SimpleText := 'Fehler in: ' + IntToStr(i);
      sleep(100);
    end;
  end;
finally
  aSqlQuery.free;
end;
```

Unterer Button Zum Unterschied vorher, ist diese Query zur Designzeit auf dem Formular hinterlegt worden. Damit entfällt das erzeugen zur Laufzeit. Das soll auch den Unterschied zwischen den beiden Methoden zeigen.

Es wird zuerst die Query inaktiv gemacht, falls sie offen war. Dann ganz einfach der Inhalt der Strings aus dem Meno in die Query übertragen und die Query wieder aktiv gemacht. Deshalb aktiv gemacht, da wir ja eine Ergebnismenge erwarten. Das hätten wir beim ausführen von *ExecSQL* nicht.

```
if SQLQuery1.Active then SQLQuery1.Active := false;
SQLQuery1.SQL.Clear;
SQLQuery1.SQL.Assign(Memo1.Lines);
try
  SQLQuery1.Active := true;
except
  showmessage('Anweisungen nicht gültig');
end;
```

Dann kann man im Datengitter die Ergebnisse betrachten.

SVN Den Quelltext des Beispiel kann man sich auch mit folgenden Kommando aus dem SVN ind das aktuelle Verzeichnis holen.

```
svn co https://lazsnippets.svn.sourceforge.net/svnroot/lazsnippets/\
  trunk/datenbank/MySQL/MySQLTestData .
```

Die Zeile ist in einem zu schreiben und wurde nur aus Formatierungsgründen hier beim Rückstrich (Backslash) umgebrochen. Der Punkt am Ende der Zeile ist notwendig, da es das Kennzeichen für das aktuelle Verzeichnis ist. Weiter ist auf die Schreibweise zu achten, das hier die Server zwischen Großbuchstaben und Kleinbuchstaben unterscheiden.

Betriebssystem	getestet
Linux Suse	nein
WinXP	ja

Abbildung 4.3: Versionsübersicht

Version `Version: $LastChangedRevision: 38 $` [5]

[5] Autor: Andreas Frieß
Lizenz: GFDL

4.2 DatenbankenSQLite3x

5 Programme

5.1 Nützliche Werkzeuge

5.1.1 Versionskontrolle

Warum ein Versionsmanagment überhaupt verwenden? Es ist doch viel einfacher den Sourcecode und die benötigten Dateien einfach auf der Festplatte zu haben und von Zeit zu Zeit macht man davon ein Archiv. Diesem Archiv gebe ich ganz einfach einen guten Titel und die Sache hat sich.

Das kann bei einem einzelnen Entwickler durchaus so noch funktionieren. Wenn man aber in einem Team arbeiten muß, so wird es schon hier zu Problemen kommen. Denn jeder Entwickler hat seine eigene Version und muß die Änderungen in das Projekt ein pflegen. Was ist aber wenn gerade 2 Entwickler an derselben Datei arbeiten. Der eine bringt die Änderungen ein, der andere bekommt die Änderung aber auch nicht mit und bringt seinerseits seine Änderungen ein. Somit sind die Änderungen des ersten Entwicklers unbemerkt verloren und das Projekt somit nicht konsistent. Weiters ist die Verfolgung der Änderungen fast, bzw. nur schwer möglich. Um dieses Szenario zu vermeiden und die arbeit im Team zu vereinfachen haben sich im laufe der Jahre Versionsmanagmentsysteme entwickelt. Da wir hier nur SVN einsetzen werden, will ich mich hier auf die grundlegende Bedienung von SVN beschränken.

svn Kommandozeile

SVN von der Kommandozeile aus zu bedienen ist eine der universellsten Möglichkeiten, das diese Art damit zu arbeiten auf allen Plattformen (Unix, Linux, Windows, MacOS, ...) vorhanden ist. Vor allen wenn man nur Projekte aus dem SVN holt und auffrischt ist die Syntax relativ einfach. Die dafür notwendigen Befehle sind hier kurz vorgestellt.

svn checkout URL [Pfad] Mit diesem Kommando wird von der URL der gespeicherte Inhalt aus dem Repository geholt und an der Stelle gespeichert die durch den Pfad angegeben ist. Fehlt die Angabe des Pfades, so wird das aktuelle Verzeichnis genommen.

svn co URL [Pfad] ist die Kurzform von „checkout“, Erklärung siehe dort.

svn help Gibt die Onlinehilfe zu SVN aus.

svn list URL und

svn list Pfad Zeigt zusätzliche Informationen zur URL beziehungsweise zum Pfad an.

svn revert Pfad Entfernt die Änderungen und stellt einen unveränderten Zustand wieder her.

svn log URL und

svn log Pfad Zeigt die Loginformationen zur URL beziehungsweise zum Pfad an.

```
Version: $LastChangedRevision: 38 $ [1]
```

[1] Autor: Andreas Frieß
Lizenz: GFDL

5.1.2 Tool Versionenselektierer

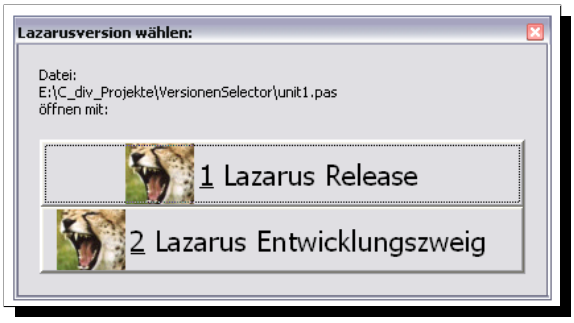
Was macht das Tool eigentlich?

Alle Betriebssysteme stellen ja die jedem bekannte Möglichkeit zur Verfügung, Dateien mit den entsprechenden Anwednungen zu verknüpfen. Somit ist es möglich mit (Doppel)Klick auf eine Datei automatisch ein entsprechendes Programm zur Bearbeitung bzw. Betrachtung zu öffnen. Und genau bei diesem 'ein' Programm liegt der Grund, wieso ich dieses kleine Tool für mich geschrieben habe. Lazarus erstellt bei der Installation wenn man es aktiviert hat, automatisch Verknüpfungen für die entsprechenden dazugehörigen Dateitypen, wie .pas, .pp, .lpr usw.

Nun ist das ja gut und schön, aber wenn man mehrer Lazarusinstallationen auf dem System laufen hat, öffnet sich ja dennoch nur die verknüpfte Version automatisch, währenddessen die andere ignoriert wird. Mehrer Lazarusversionen auf einem System zu haben kann beispielsweise den Vorteil bringen, das man sowohl das letzte offizielle Release sowie die aktuelle SVN-Version bzw. einen aktuelleren Snapshot installiert hat. Somit kann man beispielsweise die Kompatibilität der eigenen Anwednung mit neuern lazarusversionen im Auge behalten bzw. aktuelle Bugentwicklungen verfolgen, während dessen man stehts noch ein 'altes' Lazarus zum sicheren Arbeiten hat.

Wie bereits gesagt, kann ja nur eine Lazarusversion in diesem Fall zum automatischen Öffnen verknüpft werden. Möchte man allerdings die andere Nutzen muss man erst umständlich die IDE aufrufen und anschließend kann man die gewünschte Datei über 'Datei öffnen...' in die IDE laden.

Um dies zu vereinfachen und komfortabel eine Versionenauswahl zur Verfügung (siehe Screenshot) zu haben dient dieses kleine Tool. Statt mit Lazarus selbst werden die entsprechenden Dateien über die 'Öffnen mit' - Eigenschaft mit diesem Tool verknüpft. Das Programm bietet nun alle Eingetragenen Lazarusversionen zum öffnen an. Wird eine entsprechende Version ausgewählt, so startet das Tool automatisch die entsprechende Version und läd die gewünschte Datei genau so, als wenn die Datei direkt mit Lazarus verknüpft wäre.



Noch etwas in eigener Sache: das Programm wurde lediglich als kleines Hilfstool programmiert, ohne auf besondere Funktionen wert zu legen. Daher erfolgt beispielsweise die Konfiguration auch direkt in der Textdatei, wie im Folgenden erklärt wird. Jedem steht es natürlich frei, weitere Funktionalität einzubauen, ein kleiner Patch/bzw. eine Mail mit aktuellem Code dazu wäre natürlich nett. http://www.lazarusforum.de/viewtopic.php?p=11822

Konfiguration

Systemkonfiguration Das Programm selbst muss lediglich mit zusammen mit der config.ini und optional (möchte man das Icon auf den Buttons haben) mit der lazarus.bmp in ein Verzeichnis kopiert werden. Auf die Systemkonfiguration soll hier nur kurz eingegangen werden. Es ist wichtig, das die entsprechenden Dateien anstatt mit Lazarus direkt mit dem Programm verknüpft sind, so dass dieses bei öffnen der Datei gestartet wird.

Programmkonfiguration Als Konfiguration dient eine einfache Ini mit folgendem Aufbau:

```
[Pfade]
Starter=c:\lazarus\Starter\

#Die einzelnen Schaltflächen:

[Button1]
Name=Lazarus Release
Glyph=lazarus.bmp
Pfad=c:\lazarus\Lazarus.exe

[Button2]
Name=Lazarus Entwicklungszweig
Glyph=lazarus.bmp
Pfad=c:\Lazarus_trunk\Lazarus.exe
```

Allgemeine Einstellungen:

- **Starter** - diese Angabe enthält den absoluten Pfad zum Programmverzeichnis

Einstellungen für die einzelnen Schaltflächen:

- **ButtonX** - X wird für jeden neuen Button ainfach um 1 hochgezählt, ansonsten kann der gesamte Block kopiert werden und die folgenden Angaben werden angepasst:
- **Name** - Angabe, welche auf dem Button erscheint
- **Glyph** - Name des Bildes welches auf dem Button angezeigt werden soll, das Bild sollte im Programmpfad liegen
- **Pfad** - absoluter Pfad zur entsprechenden Lazarusversion, welche über diesen Button gestartet werden soll

Sind alle Konfigurationen entsprechend angepasst, ist das Programm einsatzbereit. Wird es direkt gestartet, erlaubt es ebenfalls die Auswahl und das starten einer Lazarusversion, ohne das eine Datei übergeben wird.

6 Anhang

6.1 Tabellenverzeichnis

Tabellenverzeichnis

Literaturverzeichnis

[LazDe] Deutsches Lazarusforum *http://www.lazarusforum.de* Forum

[LazEn] Englisches Lazarusforum *http://www.lazarus.freepascal.org*

[FPCLangRef] Reference guide for Free Pascal, version 2.0.4,Document version 2.0 August 2006 *ftp://ftp.freepascal.org/pub/fpc/docs-pdf/ref.pdf*

[10] Lazarus Wiki *http://www.lazarus.freepascal.org* Übersichtstabelle_der_IDE_Tastenkombinationen

[11] Homepage *http://www.ullihome.de/*

[12] Homepage *http://homepages.codegear.com/jedi/jcl/*

[14] Homepage *http://www.????.???/*

6.2 GNU Free Documentation License

Preamble

The purpose of this License is to make a manual, textbook, or other functional and useful document "free" in the sense of freedom: to assure everyone the effective freedom to copy and redistribute it, with or without modifying it, either commercially or noncommercially. Secondarily, this License preserves for the author and publisher a way to get credit for their work, while not being considered responsible for modifications made by others.

This License is a kind of "copyleft", which means that derivative works of the document must themselves be free in the same sense. It complements the GNU General Public License, which is a copyleft license designed for free software.

We have designed this License in order to use it for manuals for free software, because free software needs free documentation: a free program should come with manuals providing the same freedoms that the software does. But this License is not limited to software manuals; it can be used for any textual work, regardless of subject matter or whether it is published as a printed book. We recommend this License principally for works whose purpose is instruction or reference.

1. APPLICABILITY AND DEFINITIONS

This License applies to any manual or other work, in any medium, that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. Such a notice grants a world-wide, royalty-free license, unlimited in duration, to use that work under the conditions stated herein. The "**Document**", below, refers to any such manual or work. Any member of the public is a licensee, and is addressed as "**you**". You accept the license if you copy, modify or distribute the work in a way requiring permission under copyright law.

A "**Modified Version**" of the Document means any work containing the Document or a portion of it, either copied verbatim, or with modifications and/or translated into another language.

A "**Secondary Section**" is a named appendix or a front-matter section of the Document that deals exclusively with the relationship of the publishers or authors of the Document to the Document's overall subject (or to related matters) and contains nothing that could fall directly within that overall subject. (Thus, if the Document is in part a textbook of mathematics, a Secondary Section may not explain any mathematics.) The relationship could be a matter of historical connection with the subject or with related matters, or of legal, commercial, philosophical, ethical or political position regarding them.

The "**Invariant Sections**" are certain Secondary Sections whose titles are designated, as being those of Invariant Sections, in the notice that says that the Document is released under this License. If a section does not fit the above definition of Secondary then it is not allowed to be designated as Invariant. The Document may contain zero Invariant Sections. If the Document does not identify any Invariant Sections then there are none.

The "**Cover Texts**" are certain short passages of text that are listed, as Front-Cover Texts or Back-Cover Texts, in the notice that says that the Document is released under this License. A Front-Cover Text may be at most 5 words, and a Back-Cover Text may be at most 25 words.

A "**Transparent**" copy of the Document means a machine-readable copy, represented in a format whose specification is available to the general public, that is suitable for revising the document straightforwardly with generic text editors or (for images composed of pixels) generic paint programs or (for drawings) some widely available drawing editor, and that is suitable for input to text formatters or for automatic translation to a variety of formats suitable for input to text formatters. A copy made in an otherwise Transparent file format whose markup, or absence of markup, has been arranged to thwart or discourage subsequent modification by readers is not Transparent. An image format is not Transparent if used for any substantial amount of text. A copy that is not "Transparent" is called "**Opaque**".

Examples of suitable formats for Transparent copies include plain ASCII without markup, Texinfo input format, LaTeX input format, SGML or XML using a publicly available DTD, and standard-conforming simple HTML, PostScript or PDF designed for human modification. Examples of transparent image formats include PNG, XCF and JPG. Opaque formats include proprietary formats that can be read and edited only by proprietary word processors, SGML or XML for which the DTD and/or processing tools are not generally available, and the machine-generated HTML, PostScript or PDF produced by some word processors for output purposes only.

The "**Title Page**" means, for a printed book, the title page itself, plus such following pages as are needed to hold, legibly, the material this License requires to appear in the title page. For works in formats which do not have any title page as such, "Title Page" means the text near the most prominent appearance of the work's title, preceding the beginning of the body of the text.

A section "**Entitled XYZ**" means a named subunit of the Document whose title either is precisely XYZ or contains XYZ in parentheses following text that translates XYZ in another language. (Here XYZ stands for a specific section name mentioned below, such as "**Acknowledgements**", "**Dedications**", "**Endorsements**", or "**History**".) To "**Preserve the Title**" of such a section when you modify the Document means that it remains a section "Entitled XYZ" according to this definition.

The Document may include Warranty Disclaimers next to the notice which states that this License applies to the Document. These Warranty Disclaimers are considered to be included by reference in this License, but only as regards disclaiming warranties: any other implication that these Warranty Disclaimers may have is void and has no effect on the meaning of this License.

2. VERBATIM COPYING

You may copy and distribute the Document in any medium, either commercially or noncommercially, provided that this License, the copyright notices, and the license notice saying this

License applies to the Document are reproduced in all copies, and that you add no other conditions whatsoever to those of this License. You may not use technical measures to obstruct or control the reading or further copying of the copies you make or distribute. However, you may accept compensation in exchange for copies. If you distribute a large enough number of copies you must also follow the conditions in section 3.

You may also lend copies, under the same conditions stated above, and you may publicly display copies.

3. COPYING IN QUANTITY

If you publish printed copies (or copies in media that commonly have printed covers) of the Document, numbering more than 100, and the Document's license notice requires Cover Texts, you must enclose the copies in covers that carry, clearly and legibly, all these Cover Texts: Front-Cover Texts on the front cover, and Back-Cover Texts on the back cover. Both covers must also clearly and legibly identify you as the publisher of these copies. The front cover must present the full title with all words of the title equally prominent and visible. You may add other material on the covers in addition. Copying with changes limited to the covers, as long as they preserve the title of the Document and satisfy these conditions, can be treated as verbatim copying in other respects.

If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the actual cover, and continue the rest onto adjacent pages.

If you publish or distribute Opaque copies of the Document numbering more than 100, you must either include a machine-readable Transparent copy along with each Opaque copy, or state in or with each Opaque copy a computer-network location from which the general network-using public has access to download using public-standard network protocols a complete Transparent copy of the Document, free of added material. If you use the latter option, you must take reasonably prudent steps, when you begin distribution of Opaque copies in quantity, to ensure that this Transparent copy will remain thus accessible at the stated location until at least one year after the last time you distribute an Opaque copy (directly or through your agents or retailers) of that edition to the public.

It is requested, but not required, that you contact the authors of the Document well before redistributing any large number of copies, to give them a chance to provide you with an updated version of the Document.

4. MODIFICATIONS

You may copy and distribute a Modified Version of the Document under the conditions of sections 2 and 3 above, provided that you release the Modified Version under precisely this License, with the Modified Version filling the role of the Document, thus licensing distribution and modification of the Modified Version to whoever possesses a copy of it. In addition, you must do these things in the Modified Version:

A. Use in the Title Page (and on the covers, if any) a title distinct from that of the Document, and from those of previous versions (which should, if there were any, be listed in the

History section of the Document). You may use the same title as a previous version if the original publisher of that version gives permission.

B. List on the Title Page, as authors, one or more persons or entities responsible for authorship of the modifications in the Modified Version, together with at least five of the principal authors of the Document (all of its principal authors, if it has fewer than five), unless they release you from this requirement.

C. State on the Title page the name of the publisher of the Modified Version, as the publisher.

D. Preserve all the copyright notices of the Document.

E. Add an appropriate copyright notice for your modifications adjacent to the other copyright notices.

F. Include, immediately after the copyright notices, a license notice giving the public permission to use the Modified Version under the terms of this License, in the form shown in the Addendum below.

G. Preserve in that license notice the full lists of Invariant Sections and required Cover Texts given in the Document's license notice.

H. Include an unaltered copy of this License.

I. Preserve the section Entitled "History", Preserve its Title, and add to it an item stating at least the title, year, new authors, and publisher of the Modified Version as given on the Title Page. If there is no section Entitled "History" in the Document, create one stating the title, year, authors, and publisher of the Document as given on its Title Page, then add an item describing the Modified Version as stated in the previous sentence.

J. Preserve the network location, if any, given in the Document for public access to a Transparent copy of the Document, and likewise the network locations given in the Document for previous versions it was based on. These may be placed in the "History" section. You may omit a network location for a work that was published at least four years before the Document itself, or if the original publisher of the version it refers to gives permission.

K. For any section Entitled "Acknowledgements" or "Dedications", Preserve the Title of the section, and preserve in the section all the substance and tone of each of the contributor acknowledgements and/or dedications given therein.

L. Preserve all the Invariant Sections of the Document, unaltered in their text and in their titles. Section numbers or the equivalent are not considered part of the section titles.

M. Delete any section Entitled "Endorsements". Such a section may not be included in the Modified Version.

N. Do not retitle any existing section to be Entitled "Endorsements" or to conflict in title with any Invariant Section.

O. Preserve any Warranty Disclaimers.

If the Modified Version includes new front-matter sections or appendices that qualify as Secondary Sections and contain no material copied from the Document, you may at your option designate some or all of these sections as invariant. To do this, add their titles to the list of Invariant Sections in the Modified Version's license notice. These titles must be distinct from any other section titles.

You may add a section Entitled "Endorsements", provided it contains nothing but endorsements of your Modified Version by various parties–for example, statements of peer review or that the text has been approved by an organization as the authoritative definition of a standard.

You may add a passage of up to five words as a Front-Cover Text, and a passage of up to 25 words as a Back-Cover Text, to the end of the list of Cover Texts in the Modified Version. Only one passage of Front-Cover Text and one of Back-Cover Text may be added by (or through arrangements made by) any one entity. If the Document already includes a cover text for the same cover, previously added by you or by arrangement made by the same entity you are acting on behalf of, you may not add another; but you may replace the old one, on explicit permission from the previous publisher that added the old one.

The author(s) and publisher(s) of the Document do not by this License give permission to use their names for publicity for or to assert or imply endorsement of any Modified Version.

5. COMBINING DOCUMENTS

You may combine the Document with other documents released under this License, under the terms defined in section 4 above for modified versions, provided that you include in the combination all of the Invariant Sections of all of the original documents, unmodified, and list them all as Invariant Sections of your combined work in its license notice, and that you preserve all their Warranty Disclaimers.

The combined work need only contain one copy of this License, and multiple identical Invariant Sections may be replaced with a single copy. If there are multiple Invariant Sections with the same name but different contents, make the title of each such section unique by adding at the end of it, in parentheses, the name of the original author or publisher of that section if known, or else a unique number. Make the same adjustment to the section titles in the list of Invariant Sections in the license notice of the combined work.

In the combination, you must combine any sections Entitled "History" in the various original documents, forming one section Entitled "History"; likewise combine any sections Entitled "Acknowledgements", and any sections Entitled "Dedications". You must delete all sections Entitled "Endorsements".

6. COLLECTIONS OF DOCUMENTS

You may make a collection consisting of the Document and other documents released under this License, and replace the individual copies of this License in the various documents with a single copy that is included in the collection, provided that you follow the rules of this License for verbatim copying of each of the documents in all other respects.

You may extract a single document from such a collection, and distribute it individually under this License, provided you insert a copy of this License into the extracted document, and follow this License in all other respects regarding verbatim copying of that document.

7. AGGREGATION WITH INDEPENDENT WORKS

A compilation of the Document or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, is called an "aggregate" if the copyright resulting from the compilation is not used to limit the legal rights of the compilation's users beyond what the individual works permit. When the Document is included in an aggregate, this License does not apply to the other works in the aggregate which are not themselves derivative works of the Document.

If the Cover Text requirement of section 3 is applicable to these copies of the Document, then if the Document is less than one half of the entire aggregate, the Document's Cover Texts may be placed on covers that bracket the Document within the aggregate, or the electronic equivalent of covers if the Document is in electronic form. Otherwise they must appear on printed covers that bracket the whole aggregate.

8. TRANSLATION

Translation is considered a kind of modification, so you may distribute translations of the Document under the terms of section 4. Replacing Invariant Sections with translations requires special permission from their copyright holders, but you may include translations of some or all Invariant Sections in addition to the original versions of these Invariant Sections. You may include a translation of this License, and all the license notices in the Document, and any Warranty Disclaimers, provided that you also include the original English version of this License and the original versions of those notices and disclaimers. In case of a disagreement between the translation and the original version of this License or a notice or disclaimer, the original version will prevail.

If a section in the Document is Entitled "Acknowledgements", "Dedications", or "History", the requirement (section 4) to Preserve its Title (section 1) will typically require changing the actual title.

9. TERMINATION

You may not copy, modify, sublicense, or distribute the Document except as expressly provided for under this License. Any other attempt to copy, modify, sublicense or distribute the Document is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

10. FUTURE REVISIONS OF THIS LICENSE

The Free Software Foundation may publish new, revised versions of the GNU Free Documentation License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns. See http://www.gnu.org/copyleft/.

Each version of the License is given a distinguishing version number. If the Document specifies that a particular numbered version of this License "or any later version" applies to it, you have the option of following the terms and conditions either of that specified version or of any later version that has been published (not as a draft) by the Free Software Foundation. If the Document does not specify a version number of this License, you may choose any version ever published (not as a draft) by the Free Software Foundation.

ADDENDUM: How to use this License for your documents

To use this License in a document you have written, include a copy of the License in the document and put the following copyright and license notices just after the title page:

> Copyright © YEAR YOUR NAME. Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.2 or any later version published by the Free Software Foundation; with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts. A copy of the license is included in the section entitled "GNU Free Documentation License".

If you have Invariant Sections, Front-Cover Texts and Back-Cover Texts, replace the "with ... Texts." line with this:

> with the Invariant Sections being LIST THEIR TITLES, with the Front-Cover Texts being LIST, and with the Back-Cover Texts being LIST.

If you have Invariant Sections without Cover Texts, or some other combination of the three, merge those two alternatives to suit the situation.

If your document contains nontrivial examples of program code, we recommend releasing these examples in parallel under your choice of free software license, such as the GNU General Public License, to permit their use in free software.