LivingXML IVI::DB 0.7

Handbuch

1. Was ist die IVI::DB ?

Die IVI::DB ist eine native XML Datenbank, das heißt, daß man darin XML Dokumente direkt abspeichern, und wiederverwenden kann.

Die Datenbank ist darauf ausgelegt, eine große Anzahl (Millionen, Milliarden) von kleinen bis mittleren (10 Bytes bis 10 KB) Datensätzen zu verwalten, indizieren und verfügbar zu machen.

Die wichtigsten Ziele bei der Entwicklung der Datenbank waren:
- Performance
- Flexibilität
- Einfach und klein

1.1. Vorteile

- Logische Datensätze müssen nicht auf mehrere Tabellen verteilt werden, was Performance und Nerven kostet
- XML Datensätze müssen nicht in eine relationale oder objektorientierte Struktur konvertiert werden.
- Flexible, transparente Indizierung
- Visualisierungen von Datensätzen können in der Datenbank verwaltet werden, und können mit der Performance von statischen Seiten serviert werden.
- Der Datenbankkern ist extrem klein, und kann komplett in andere Applikationen integriert sein.
- Komplette State-of-the-Art Applikationen können inklusive der Datenbankengine wieder auf 3,5" Disketten ausgeliefert werden
- Struktur-unabhängigkeit: Sie können gleich loslegen, und Ihre Daten in die Datenbank einspielen, ohne sich vorher Gedanken über deren Struktur machen zu müssen.
- Die Datenbank kann leicht an eigene Anforderungen angepasst werden, und dadurch auch extreme Aufgaben erfüllen (von einer reinen Suchmaschine ohne Updates, über normalen Datenbanken bis zur überwiegenden Datensenke Logging)
- Unicode Unterstützung (die Daten werden intern in UTF-8 verwaltet)
- Die Datenbank unterliegt der GPL Lizenz

1.2. Nachteile

- Stark relationale, nicht-hierarchische Daten mit fixer Struktur sind wahrscheinlich schlecht verwaltbar.
- Komplexe Abfragen, numerische Abfragen, sind derzeit nicht in der Indizierungsengine verfügbar. Ausweg: Komplexe Abfragen mit XSLT im Output-Stylesheet lösen
- Die Datenbank läuft nicht in vollem Umfang unter Windows (symbolische Links sind für die Indizierung notwendig)
- Große Datensätze können zu Performanceproblemen führen, da derzeit noch mit temporärem DOM gearbeitet wird. Empfehlung: Große Datensätze in Tabellen mit mehreren kleinen Datensätzen darin zu verwandeln. Der geplante Ausweg sind SAX basierte Transformationen und persistente DOM Dateisysteme.

2. Installation

2.1. Systemanforderungen

Notwendig:
- Apache Webserver (www.apache.org)
- Sablotron (www.gingerall.com)
- Perl (www.perl.org)
- XML::Sablotron (www.gingerall.com)
- File::CounterFile (www.cpan.org)
- File::Path (www.cpan.org)
- Dateisystem das symlinks unterstützt (ReiserFS, Ext2, Ext3, XFS, JFS, ...)

Empfohlen:
- mod_perl (www.modperl.org)
- ReiserFS als Dateisystem (www.reiserfs.org)
- SuSE Linux 7.3/8.0 (www.suse.com)
- XSLT Kenntnisse (selfhtml.teamone.de)
- Text::Soundex (www.cpan.org)

2.2. Installationsvorgang

IVI::DB wird einfach in das htdocs Verzeichnis und das cgi-bin Verzeichnis des Webservers installiert.

Sodann steht IVI::DB unter

```
http://server/cgi-bin/ivi?_style=ivi.xsl
```

die Administrationsoberfläche zur Verfügung.

3. Erste Schritte

3.1. IVI Tour

Starten Sie ihren Browser, und gehen sie zu:

```
http://server/ivitour.htm
```

Dort klicken Sie sich bitte langsam der Reihe nach durch die Zahlen durch.

3.2. Arbeiten mit der Datenbank

Die grundsätzliche (empfohlene) Vorgangsweise zum Umgang mit der Datenbank ist folgende:

Anlegen einer Tabelle (Admin)
Einspielen eines oder mehrerer Datensätze (Admin)

Anlegen der Indizierung (Admin)
Anlegen der Visualisierung (Admin)
Abfragen von Daten (Admin)
Anbinden eigener Applikationen

Diese Vorgehensweise hat folgende Gründe:

Für das Einspielen von Datensätzen ist es nicht notwendig der Datenbank die genaue Struktur der Daten mitzuteilen.
Wenn bereits Datensätze vorhanden sind, kann deren Struktur automatisch für die Erstellung der Indizes und der Visualisierungen verwendet werden.

Nachdem der Admin komplett auf die Datenbankschnittstelle aufbaut, brauchen Sie für Ihre eigene Applikation nur die URL aus der Adminoberfläche hernehmen, und den Teil
&_style=ivi.xsl
entfernen, oder ihn durch ein eigenes Stylesheet ersetzen.

Um eine Person zu suchen, kommen Sie mit der Admin Oberfläche zum Beispiel zu diesem Ergebnis:

```
http://www.livingxml.net/cgi-bin/ivi?_path=rt.sound&_index=1&PERSON/VORN
AME=robart&_style=ivi.xsl
```

Um dasselbe Ergebnis in purem XML für Ihre Anwendung zu bekommen, entfernen Sie den _style Parameter:

```
http://www.livingxml.net/cgi-bin/ivi?_path=rt.sound&_index=1&PERSON/VORN
AME=robart
```

4. Grundkonzept

Hinter der IVI::DB stehen eine ganze Reihe von Konzepten, die aus beobachteten Mängeln und Vorteilen anderer Datenverwaltungssysteme entstanden sind.

4.1. Logische Datensätze

Sogenannte Logische Datensätze sind Datensätze, die eigentlich zusammengehören, und bei relationalen Datenbanken durch die Normalisierung in mehrere physische Datensätze aufgespalten werden.
Durch die Aufspaltung von eigentlich atomar zu behandelnden Datensätzen entstanden eine Reihe von Problemen:

- Man braucht jedesmal eine Menge an Code auf Anwendungsseite, um die Aufspaltung und Wieder-Zusammenführung zu bewerkstelligen
- Die Aufspaltung ist auf der Ebene der SQL Schnittstelle fast zwingend vorgeschrieben, und macht daher die SQL Schnittstelle zum Flaschenhals
- Man muß Konzepte wie Transaktionen, Referentielle Integrität, ... mißbrauchen, um die Atomare Behandlung der Logischen Datensätze sicherzustellen. Dies kostet massiv Performance.

Eines der obersten Ziele der IVI::DB ist es daher, die Möglichkeit zur Verfügung zu stellen, Logische Datensätze als solches atomar verwalten zu können, ohne sie aufspalten zu müssen. Dies wird dadurch erreicht, daß logische Datensätze als komplette XML Dokumente repräsentiert sind, und in sich hierarchisch strukturiert sind.

4.2. Hierarchisches Verwaltungssystem

XML ist in sich hierarchisch strukturiert. Es macht daher wenig Sinn, XML zum Beispiel in Tabellenform abzuspeichern, da Tabellen wenig geeignet sind, komplexe hierarchische Strukturen zu verwalten.

4.3. Tabellen, Untertabellen

Bei mandantenfähigen Anwendungen steht man schnell an den Grenzen herkömmlicher Datenverwaltungssysteme. Speichert man die Daten unterschiedlicher Mandanten in derselben Tabelle? Verwendet man für jeden Mandanten eine eigene Datenbank?
Viele Anwendungen aus der Praxis verwenden ihre eigenen Systemtabellen "Einstellungen", die nur einen Datensatz enthalten, in dem sie zentral alle Ihre Einstellungen abspeichern.

Wir haben uns von diesem Konzept getrennt, und versuchen mit der IVI::DB ein hierarchisches Tabellenkonzept:

- Datenbanken sind Tabellen.
- Tabellen können Tabellen und Datensätze enthalten.

```
db/

db/Mandant1/

db/Mandant1/EigeneAdresse.xml

db/Mandant1/Einstellungen.xml
```

```
db/Mandant1/Adressen/
db/Mandant1/Adressen/Kunde1.xml
db/Mandant1/Adressen/Kunde2.xml
db/Mandant1/Adressen/Kunde3.xml
db/Mandant1/Auftraege/
db/Mandant1/Auftraege/Bestellung1.xml
db/Mandant1/Auftraege/Bestellung2.xml
db/Mandant1/Auftraege/Bestellung3.xml
db/Mandant2/
db/Mandant2/EigeneAdresse.xml
db/Mandant2/Einstellungen.xml
db/Mandant2/Adressen/
db/Mandant2/Adressen/Kunde1.xml
db/Mandant2/Adressen/Kunde2.xml
db/Mandant2/Auftraege/
db/Mandant2/Auftraege/Bestellung1.xml
db/Mandant2/Auftraege/Bestellung2.xml
```

Die Integrität der Relationen wird automatisch durch die hierarchische Struktur gewährleistet. Eine Relations-Engine, die nicht-hierarchische Relationen absichert ist in Planung.

4.4. Primärschlüssel

Für die Datenbank besteht ein Datensatz aus 2 Dingen: Dem Primärschlüssel, und dem Dateninhalt. Das heißt, der Primärschlüssel ist nicht Teil der Daten, und kann daher beliebig aus den Daten generiert werden, oder auch von den Daten unabhängig sein, das ist der Applikation überlassen.

Beim Einlagern eines Datensatzes in die Datenbank kann von der Anwendung/vom Benutzer ein Primärschlüssel mitgegeben werden. Wenn dieser mitgegeben wird, dann wird der Datensatz unter diesem Primärschlüssel gespeichert. Wenn kein Primärschlüssel mitgegeben wird, dann wird von der Datenbank automatisch ein neuer Schlüssel erzeugt, und zurückgegeben.

Beispiele:

```
./ivi _path=rt
_process=<PERSON><VORNAME>Philipp</VORNAME><FAMNAME>Maier</FAMNAME></PER
```

```
SON>
```

legt den Personendatensatz ab, und generiert automatisch eine ID dafür.

```
./ivi _path=rt _id=20020602
_process=<LOG><DATUM>20020602</DATUM><Wert>125</WERT></LOG>
```

legt den Log Eintrag unter der ID "20020602" ab.

4.5. Indizierung

Viele andere Datenverarbeitungssysteme verwenden transparente Indizierung. Das heißt, daß man einen Index definiert (Alle Vornamen als Text indizieren), bei den Abfragen aber allgemeine Anfragen stellt ("suche alle Datensätze, bei denen der Vorname 'Philipp' ist").
Dies hat den Nachteil, daß die Datenbankengine die allgemeine Anfrage parsen muß, und erraten muß, mit welchem der definierten Indizes die Anfrage wohl am besten zu beantworten ist.
Daher verwenden wir derzeit nur Nicht-Transparente Indizierung:
Man definiert einen Index (Der Index Adressen.Vornamen indiziert von allen Adressen die Vornamen als Text)
Und sucht dann über den Index ("Suche im Index 'Adressen.Vornamen' nach dem Vornamen 'Philipp' ")
Dadurch ist die Suchperformance für die Anwendung definiert, einschätzbar und nicht von der Lust und Laune einer Engine abhängig. (Stichwort: Full Table Scan)

Indizes haben die Namenskonvention Tabelle.Index (mit Punkt getrennt)
Dies ist notwendig, weil beim Ändern von Daten in einer Tabelle allen Tabelle.* die Indizierung und die Visualisierung mitgezogen werden müssen.

Indiziert werden können normalerweise Textelemente (<Vorname>Philipp</Vorname>) und Attribute von XML Elementen (<feld typ="bunt"/>)

Zusätzlich liefert die Datenbankengine bei einer normalen Suchabfrage (die analysiert und kompiliert werden muß) zum Suchergebnis auch noch die kompilierte Form der Suchabfrage mit, damit weitere Suchabfragen noch schneller durchgeführt werden können.

Als Wildcards für die Suche stehen
- * für beliebig viele Zeichen
- ? für genau ein Zeichen

zur Verfügung.

4.5.1. Interne Realisierung

Die Indizierung wird intern folgendermaßen realisiert:
Alle Datensätze einer Tabelle werden als XML Dateien in ein Verzeichnis gelegt.

```
Adressen/1      (Philipp Guhring)
Adressen/2      (Franz Wieser)
```

```
Adressen/3      (Philipp Huber)
...
```

Um alle Vornamen zu indizieren wird ein Index Verzeichnis angelegt, in dem für jeden Datensatz ein symbolischer Link existiert.
Der Name jeden symbolischen Links ist der zu indizierende Wert.
Das Ziel des symbolischen Links ist der Datensatz:

```
Adressen.Vorname/Philipp_1 -> ../Adressen/1
Adressen.Vorname/Franz_2 -> ../Adressen/2
Adressen.Vorname/Philipp_3 -> ../Adressen/3
```

Wenn nun nach dem Vornamen Philipp gesucht wird, ist folgende Suchabfrage notwendig:
cat Adressen.Vorname/Philipp_*
Da die Namen der symbolischen Links im Verzeichnis eindeutig sein müssen, werden auch noch die Primärschlüssel der Datensätze drangehängt.

4.6. Abfragen

Entsprechend der Indizierung gibt es 3 Möglichkeiten, Daten abzufragen:

• Direkte Abfrage über die ID(Primärschlüssel):

```
./ivi _path=/Adressen _xql=45
```

• Abfrage auf einen Index:
Einfache Form:

```
./ivi _path=/Adressen.Vorname _index=1 PERSON/VORNAME=Philipp
```

Kompilierte Form:

```
./ivi _path=/Adressen.Vorname "_xql=Philipp_*"
```

• Komplette Tabellen

```
./ivi _path=/Adressen _xql*
```

Das Problem an den 3 Methoden ist, daß man nur direkt suchen kann, was man indiziert hat. Man hat keine Möglichkeit, nach Feldern oder Attributen zu suchen, die nicht im Index indiziert sind, den man verwendet.

Wenn man trotzdem nach Feldern selektieren muß, die man nicht indiziert hat, muß man eine Suchabfrage definieren, die mindestens alle in Frage kommenden Datensätze zurückliefert.

Und über diese Suchabfrage läßt man dann ein Stylesheet laufen, das die restliche Selektion erledigt:

philippselect.xsl:

```
<xsl:for-each select="/ivi:response/ivi:result/ivi:data">
 <xsl:if test="/ADRESSE/VORNAME = 'Philipp'">
```

```
  <xsl:copy-of select="."/>
 </xsl:if>
</xsl:for-each>

./ivi _xql=/Adressen/* _style=philippselect.xsl
```

4.7. Visualisierung

Bei einer Visualisierung verwaltet die Datenbank zu jedem Datensatz einer Tabelle dessen fertiger Visualisierung, in HTML zum Beispiel.

Die Rohdaten:

```
<ADRESSE>
  <VORNAME>Philipp</VORNAME>
  <FAMNAME>Guhring</FAMNAME>
</ADRESSE>
```

Die Visualisierung:

```
<tr>
  <td align="right">Philipp</td>
  <td align="center">|</td>
  <td bgcolor="#e0e0ff">Guhring</td>
</tr>
```

Philipp	|	Gühring

Das dafür verwendete Stylesheet:

```
<tr>
  <td align="right"><xsl:value-of select="ADRESSE/VORNAME"/></td>
  <td align="center">|</td>
  <td bgcolor="#e0e0ff"><xsl:value-of select="ADRESSE/FAMNAME"/></td>
</tr>
```

Man kann von denselben Rohdaten mehrere verschiedene Visualisierungen machen. Eine Listenansicht und eine Detailansicht zum Beispiel.

Visualisierungen sollte man dort einsetzen, wo man eher statische Daten hat, die eher aufwändig, oder sehr oft (viele Besucher eines Portals zum Beispiel) visualisiert werden müssen.

Ein anderes mögliches Einsatzgebiet sind Berechnungen wie zum Beispiel Gesamtsummen von Angeboten und Rechnungen, wenn die im XML Dokument nicht mit enthalten sein können.

Visualisierungen können Indiziert werden:
Die Rohdaten werden indiziert, und das Ziel(Destination) der Indizierung ist die Visualisierung. Mit der Kombination bekommt man fertige dynamische Seiten mit exzellenter Performance serviert.

Eine Einschränkung des Systems ist natürlich, daß die Visualisierung eines Datensatzes nur von genau diesem abhängen kann, und sonst keine dynamik möglich ist.

5. Zugriffsarten auf die Datenbank

5.1. Direkte Anfrage:

```
_process := <root>Ein Datensatz</root>

_path := test
```

Direkte Anfragen bestehen aus einer Reihe von benannten Kommandos und Optionen. Die genauen Regeln zur Verwendung der Kommandos und Optionen

5.2. Batchmodus: (geplant)

```
<ivi:batch>

  <ivi:command _path="test" _process="<root>Ein Datensatz</root>"/>

  <ivi:command _path="test" _xql="Adressen/*"/>

</ivi:batch>
```

Der Batchmodus kann als Direkte Anfrage verschickt werden:
_batch := <ivi:batch><ivi:command/><ivi:command/></ivi:batch>

5.3. Automatische Verarbeitung: (geplant)

...Datenstrom...
<DOKUMENT>
 <Dokumentdaten>daten...daten..daten...</Dokumentdaten>
 <Daten>daten</Daten>
</DOKUMENT>

Bei der Automatischen Verarbeitung wird ein XML Dokument aus einem Datenstrom anhand des Root-Elements ("DOKUMENT") erkannt, und einer vordefinierten Verarbeitung zugeführt.

Zum Beispiel:
Wenn in einer EMail ein XML Dokument mit dem Root Element ADRESSE auftritt, wird das XML Dokument extrahiert, und in die Adressetabelle aufgenommen.

6. Zugriffsmethoden zur Datenbank

6.1. HTTP (direkte Anfragen)

6.1.1. Browser

```
http://server/cgi-bin/ivi?_style=ivi.xsl
http://server/cgi-bin/ivi?_style=ivi.xsl&_process=<root>Ein
Datensatz</root>
```

6.1.2. Webservice

```
http://server/cgi-bin/ivi?_path=/
http://server/cgi-bin/ivi?_process=<root>Ein Datensatz</root>
```

6.2. Lokal (direkte Anfragen)

IVI::DB kann direkt auf der Kommandozeile oder auch von Shellscripts oder anderen Programmen aus verwendet werden:

```
cd /web/cgi-bin
./ivi _diagnose=version
./ivi _define=test
./ivi _path=test "_process=<root>Ein Datensatz</root>"
./ivi _path=test "_process=<root>Noch ein Datensatz</root>"
./ivi _path=test "_process=<my xml='test'>root (der dritte
Datensatz)</my>"
```

6.3. IVI::DB Perl Modul (direkte Anfragen) (geplant)

```
use IVI::DB;
ivi(_path=>"test" _process=>"<root>Ein Datensatz</root>");
my $result=ivi(_path=>"test" _xql=>"Adressen/*");
```

6.4. libivi (geplant)

```
#include <ivi.h>
ivi_define("mytable");
result=ivi_xql("/Adressen.Vorname/Philipp_*");
```

6.5. XSLT Funktionen (direkte Anfragen) (geplant)

```
<xsl:value-of select="ivi("/Adressen.Vorname/Philipp_*")/*/FAMNAME"/>
```

6.6. EMail (automatische Verarbeitung) (geplant)

Über das EMail Gateway wird eine automatische Verarbeitung gemacht, daß heißt im EMail Body und in den EMail Attachements wird nach wohlgeformten XML Dokumenten gesucht.

Standardregel ist die Erkennung von ivi::batch Meldungen, die in die Batchverarbeitung der Datenbank kommen.

Möglich wäre hier auch der Mimetype application/xml .

Hier ein Beispiel:

```
Von: Philipp Guhring <pg@futureware.at>
An: Datenbank <ividb@livingmxl.net>

Liebe Datenbank,
Konntest du bitte folgende Adresse automatisch in der Datenbank ablegen?
Ich hab leider keine Zeit, das selber zu macehn.
<ADRESSE>
  <Vorname>Philipp</Vorname>
  <Famname>Guhring</Famname>
</ADRESSE>
Vielen Dank.
Schone Gruße,
Philipp
```

Im XML-EMail Gateway wird die Regel hinterlegt, daß alle <ADRESSE> Elemente, die im Datenstrom gefunden werden direkt in die Tabelle /Adressen/ abgelegt werden. Wichtig ist hierbei, daß innerhalb von <ADRESSE> kein weiteres <ADRESSE> Element vorkommen darf.

7. Syntax der Datenbank

7.1. _xql

Macht eine direkte Suche:

```
_path := /Adressen
```

```
_xql := 45
```

Holt die Adresse mit der ID 45

```
_xql := *
```

Sucht alle Datensätze der Tabelle

```
_xql := Phil*
```

Holt alle Datensätze, bei denen die ID mit Phil beginnt

Argumente:

```
_max := 10
```

Holt die ersten 10 Suchergebnisse

```
_raw := yes
```

Gibt alle Suchergebnisse ohne Header und Footer zurück

7.2. _index

Macht eine Index-Suche, das heißt es nimmt Index-Anfragen entgegen, kompiliert diese, und macht dann eine _xql Suche.

Beispiel:

```
./ivi _path=rt.sound _index=1 "PERSON/VORNAME=robart"
```

Sucht im Index rt.sound nach einer Person bei der der Vorname nach Robärt klingt.

7.3. _process

Fügt neue Datensätze in eine Tabelle ein, oder überschreibt bestehende

7.3.1. Argumente:

```
_id
```

Das Argument _id wird verwendet, um der Datenbank die ID mitzuteilen.
Wenn keine ID mitgegeben wird, dann wird automatisch eine neue ID generiert. Die ID Generierung ist Tabellenbezogen, das heißt verschiedene Datensätze in verschiedenen Tabellen können diesselbe ID haben. Die IDs gelöschter Datensätze werden nicht wieder vergeben.
Wenn die ID bereits vorhanden ist, dann wird der Datensatz überschrieben.

7.4. _form

Verarbeitet HTML Formular Daten.

```
_form := meinstylesheet.xsl
```

Die vom HTML Formular bekommenen Daten werden (mit Ausnahme aller mit Parameter, deren Name mit "_" beginnen) in eine XML Struktur gegossen:

```
<root>
  <param name="vorname">Philipp</param>
  <param name="famname">Guhring</param>
  <param name="email">pg@futureware.at</param>
  <param name="submit">Subscribe!</param>
</root>
```

Dieses XML wird dann über das im Parameter _form angegebene Stylesheet angewendet, und das resultierende XML wird als Datensatz in die Datenbank abgelegt:

```
meinstylesheet.xsl:
<ADRESSE>
 <VORNAME><xsl:value-of select="/root/param[ @name='vorname'] /></VORNAME>
 <FAMNAME><xsl:value-of select="/root/param[ @name='famname'] </FAMNAME>
</ADRESSE>
```

meinstylesheet hat dann die Möglichkeit, die Parameter auszuwerten, und ein XML Dokument aus den Parametern zu bauen, das dann in der Datenbank (mittels _process) abgelegt wird:

```
<ADRESSE>
 <VORNAME>Philipp</VORNAME>
 <FAMNAME>Guhring</FAMNAME>
</ADRESSE>
```

7.5. _define

Legt eine neue Subtabelle oder einen Index an.
_define wird vor allen anderen Kommandos ausgeführt, damit man in einer direkten Anfrage eine Tabelle anlegen, und gleich danach mit den ersten Daten befüllen kann. _define ist also mit _form und _process kombinierbar.

Der Wert des Arguments _define ist der Name der neuen Tabelle.

Wenn zusätzlich das Argument _table mit angegeben wird, dann ist der Name der neuen Tabelle
"_table._path"

Um eine einfache neue Tabelle anzulegen:

./ivi _define=Tabelle

Um dann zu dieser Tabelle einen Index anzulegen:

./ivi _define=Index _table=Tabelle _form=iviparamindex.xsl _id=.index.xml _path=
_destination_table=Tabelle _destination_element=ID DATEN=normal

Dann existiert die Tabelle "Tabelle" und der Index "Tabelle.Index"

Die weiteren Argumente zum Anlegen eines Index:
• _form=iviparamindex.xsl
• _id=.index.xml
• _path=
Der Path legt fest, in welcher Tabelle dieser Index angelegt werden soll

• _destination_table=Tabelle
• _destination_element=ID
Das Destination Table Argument ist normalerweise der Tabellenname, und das Destination Element ist normalerweise die ID.
Man hat hier aber auch die Möglichkeit, Verweise/Referenzen zu indizieren:

Tabelle1:
Datensatz Philipp:
<name>Philipp</name>
<verweis>450</verweis>

Tabelle2:
Datensatz 450:
<daten/>

Nun indiziert man die Tabelle1, indiziert dort den Namen, läßt den Index aber statt auf den Datensatz selber auf den Verweis zeigen:

• _destination_table=Tabelle2
• _destination_element=verweis

Und schlußendlich kommen noch die eigentlich zu indizierenden Elemente:

• DATEN=normal

Dies bedeutet, daß das Feld Daten normal als Text indiziert werden soll.

• ADDRESS/NAME=soundex

bedeutet, daß das Feld ADDRESS/NAME mit dem Soundex Algorithmus indiziert wird. Dadurch kann eine Suche nach "Maier" alle "Maier", "Meier" und sogar "Mayer" automatisch finden.

• CHAPTER/TEXT=fulltext

wird eine Volltextindizierung machen. Ist derzeit noch nicht verfügbar.

• ADDRESS/COUNTRY=lowercase

Indiziert den Namen in Kleinbuchstaben, und findet "Austria", "AUSTRIA" und "austria", wenn jemand nach "AuStRiA" sucht.

7.6. _diagnose

Diagnose Funktionen:
• _diagnose := version liefert die Version der Datenbankengine
• _diagnose := ping liefert einen Pong als Antwort, wenn die Datenbank läuft
• _diagnose := echo liefert alle Umgebungsvariablen
• _diagnose := all liefert version, ping und echo

7.7. Allgemeine Argumente

7.7.1. _style

Mit dem Argument _style hat man die Möglichkeit über das Datenbankergebnis gleich ein XSLT Stylesheet laufen zu lassen.

Um die Administrationsoberfläche im Browser zu erhalten, mußt das Argument _style den Wert ivi.xsl
bekomen: http://server/cgi-bin/ivi?_style=ivi.xsl

Die .xsl Stylesheets müssen im cgi-bin Verzeichnis des Webservers liegen, um direkt verwendet werden zu können.

Mögliche Schreibweisen:
_style := ivi.xsl
_style := file://ivi.xsl
_style := http://server/directory/style.xsl (kann grundsätzlich funktionieren, die Verfügbarkeit hängt aber von der Installation ab, wird von uns nicht empfohlen.)

7.7.2. _path

Das Argument _path definiert die Tabelle, in der die anderen Aktionen ablaufen. Untertabellen werden mit / voneinander getrennt.

Beispiele:

```
./ivi _path=/
```

```
./ivi _path=/mandant1/Auftraege/
```

Wenn außer _path keine Kommandos angegeben werden, dann liefert die Datenbank eine Liste der Tabellen und Datensätze.

8. Stored Procedures

Für Stored Procedures empfehlen wir Perl.
Stored Procedures müssen im cgi-bin Verzeichnis des Webservers abgelegt werden.
Die Stored Procedures in Perl werden Zugriff auf die Datenbank-Funktionen als Perl Modul haben.
An einer genauen Trigger/Event Schnittstelle wird derzeit noch gearbeitet.

9. Syntaxdefinition der Indizes

Für die Indizierung wurden einige Erweiterungen zu XSLT definiert, die von einem integrierten DOM Parser abgehandelt werden.

Die Index Sprache sieht folgendermaßen aus:

```
<root xmlns:ivi="http://livingxml.net/2002/ivi">
 <index>
  <source><parameter name="Person/Vorname"><lowercase><xsl:value-of
select="Person/Vorname"/></lowercase></parameter>_<parameter
name="Person/Familienname"><lowercase><xsl:value-of
select="Person/Familienname"/></lowercase></parameter>_<xsl:value-of
select="\$id"/></source>
  <destination>../test/<xsl:value-of select="\$id"/></destination>
 </index>
</root>
```

Der erste Verarbeitungsschritt ist die XSLT Verarbeitung der Index Definition gegen das XML Dokument, das indiziert werden soll.

```
<root xmlns:ivi="http://livingxml.net/2002/ivi">
 <index>
  <source><parameter
name="Person/Vorname"><lowercase>Philipp</lowercase></parameter>_<parame
ter
name="Person/Familienname"><lowercase>Guhring</lowercase></parameter>_13
</source>
  <destination>../test/13</destination>
 </index>
</root>
```

Das resultierende XML muß dann ein <root> Element und darunter beliebig viele <index> Elemente enthalten.
Alle <index> Elemente enthalten 2 Teile, <source> und <destination>.
<source> definiert, welche Felder des XML Dokuments indiziert werden. (Vorname, Familienname zum Beispiel)

Die <destination> definiert, wo der Index hinzeigen soll. (Normalerweise auf den eigentlichen

XML Datensatz)
Dies ist notwendig, damit eine Indizierung von Referenzen zu anderen Tabellen möglich wird.

<source> besteht aus mehreren zusammengehängten Parametern:

Einfacher Index von Vorname und Familienname einer Adresstabelle:

```
vorname_famname_id -> ../adr/id
```

Um nach allen Personen zu suchen, mit dem Vornamen "Petr":

Auf der Kommandozeile:

```
ls Petr_*_*
cat Petr_*_*
Mit der Datenbank:
_xql=Petr_*_*
```

Der Wildcard "*" bedeutet, daß der Nachname und die ID beliebig ist.

Als eigene Funktionen stehen

```
<lowercase/>
<soundex/>
<normal/>
```

zur Verfügung

Durch die Vorverarbeitung mit XSLT kann man auch eine ganze Menge von Adressen, die in einem XML Dokument gespeichert sind einzeln indizieren:

```
<root xmlns:ivi="http://livingxml.net/2002/ivi">
 <xsl:for-each select="addresses/adress">
  <index>
   <source><parameter name="Person/Vorname"><lowercase><xsl:value-of
select="Person/Vorname"/></lowercase></parameter>_<parameter
name="Person/Familienname"><lowercase><xsl:value-of
select="Person/Familienname"/></lowercase></parameter>_<xsl:value-of
select="\$id"/></source>
```

```
   <destination>../test/<xsl:value-of select="\$id"/></destination>
  </index>
 </xsl:for-each>
</root>
```

Oder selektive Indizes:

```
<root xmlns:ivi="http://livingxml.net/2002/ivi">
 <xsl:if test="not(geloescht)">
  <index>
   <source><parameter name="Person/Vorname"><lowercase><xsl:value-of
select="Person/Vorname"/></lowercase></parameter>_<parameter
name="Person/Familienname"><lowercase><xsl:value-of
select="Person/Familienname"/></lowercase></parameter>_<xsl:value-of
select="\$id"/></source>
   <destination>../test/<xsl:value-of select="\$id"/></destination>
  </index>
 </xsl:if>
</root>
```

Indiziert nur die vorhandenen (noch nicht gelöschten) Datensätze.

10. Administration

10.1. Datenintegrität

Die Wohlgeformtheit der XML Daten können Sie mit dem Programm xmltest testen, und bei Bedarf mit einem Editor wie yudit, vi, kedit, Emacs, .. wiederherstellen.

Die Funktionalität von XSLT Stylesheets kann mit dem Programm xsltest getestet werden.

10.2. Backup & Restore

Als Backup und Restore Verfahren stehen derzeit bewährte Backup Werkzeuge von Unix zur Verfügung:
Um ein Backup zu machen können Sie folgendes tar Kommando verwenden:

```
tar cvzf backup-datum.tgz db/
```

Um ein Backup wiederherzustellen:

```
tar xvzf backup-datum.tgz
```

```
ivirestore
```

IVI-Restore (geplant) verifiziert die Integrität der restaurierten Daten und die Konsistenz der Datenbank.

Alternativ können bei Bedarf natürlich auch andere Backupwerkzeuge verwendet werden, wie Amanda, Taper, ...
Als Backupmedien empfehlen wir Festplatten, CD-ROMs und DVD. (Gründe: Verbreitungsgrad von Lesegeräten, Direkt-Zugriff)

10.3. Datenreplikation

IVI::DB bietet derzeit noch keine integrierten Methoden zur Datenreplikation an. Als möglichen Auswegen gibt es die Möglichkeit, Dateisystem-Replikationsmechanismen zu verwenden, wie SAN (Storage-Area-Networks), Intermezzo für Linux, ...

10.4. Bootup Programm (geplant)

Das bootup Programm sollte bei jedem Systemstart gestartet werden, um die committeten Transaktionen fertigzustellen, und die offenen Transaktionen zu rollbacken.

11. Beispiele

Weitere Beispiele gibt es unter http://www.livingxml.net/

12. Geplante Zusatzmodule:

Geplant ist derzeit:

- EMail-Gateway, um Daten per Email an die Datenbank schicken zu können
- Applikationsserver (Session Management, Compiled Stylesheets, User Management, Authorizations, ...)
- Workflow Manager (Workflow Design, Steuerung der Applikationen, Regressionstests mit XML Anwendungen, ...)
- TUX2 Variante, um die Datenbank-Abfragen zu beschleunigen, und zum Beispiel Visualisierungen schnell zu servieren

Wenn Sie Interesse an den geplanten Funktionen und Modulen haben, kontaktieren Sie uns bitte.

13. Support

Bei Problemen, Fragen, Wünschen, Fehlern, ... kontaktieren Sie bitte
support@livingxml.net