Kleine Helferlein

Jens Behring
its-people

Schlüsselworte:

Analytische Funktionen, Reguläre Ausdrücke, With-Klausel, Virtuelle Spalten, Infrastrukturdaten

Einleitung

Dieser Vortrag stellt Funktionalitäten vor, die sich im Rahmen der Erstellung einer Eisenbahninfrastrukturdatenbank als äußerst nützlich und unverzichtbar erwiesen haben. Weiterhin werden neue Funktionalitäten aus Oracle 11g vorgestellt, die das Potenzial haben, ebenfalls diesen Status zu erreichen.
Auch wenn jeder der oben genannten Punkte alleine für mehrere Vorträge ausreicht, werden hier alle erwähnt. Dieser Vortrag soll lediglich als Anregung dienen, sich mit diesen (gar nicht so) „kleinen Helferlein“ auseinanderzusetzen und auszuloten, wie und wo die eine oder andere vorgestellte Funktionalität genutzt werden kann. Folgende Funktionalitäten werden vorgestellt:

- Analytische Funktionen: ihnen haftet an, für DWH-Auswertungen gedacht und gemacht worden zu sein. Allerdings können diese Funktionen auch in „normalen“ Datenbanken gute Dienste leisten.
- Reguläre Ausdrücke: sie sind äußerst hilfreich, wenn alphanumerische Daten untersucht, gefiltert und bearbeitet werden müssen.
- Virtuelle Spalten: sie sind hervorragend dazu geeignet, Spalten, deren Inhalt sich aus vorhandenen Informationen errechnet, an Tabellen hinzuzufügen, ohne vorhandene Ladeprozesse anpassen zu müssen.
- With-Klausel: sie bietet die Entwicklern die Möglichkeit, komplexe Select-Statements aufzuteilen und übersichtlich zu gestalten.

Eisenbahninfrastruktur

Eisenbahninfrastruktur lässt sich nicht in wenigen Zeilen komplett erklären. Daher werden hier nur die im Vortrag genutzten Informationen erwähnt und kurz erläutert. Unter Eisenbahninfrastruktur wird all das verstanden, was benötigt wird, um Züge fahren lassen zu können. Dies sind z.B. Gleise, Weichen, Brücken, Tunnel. Jedes Objekt hat neben Attributen wie Bezeichnung oder Name eine Lage-Information bestehend aus STRECKE_NR, VON_KM und BIS_KM, wobei letztere in verschiedenen Schreibweisen vorliegen können, was durch einen entsprechenden Postfix am Spaltennamen gekennzeichnet ist. Die verschiedenen Schreibweisen werden hier allerdings nicht weiter erläutert, da sie in Bezug auf den eigentlichen Inhalt des Vortrages keinen Einfluss haben.

Analytische Funktionen

Allgemeines

Teilweise bereits in Oracle 8i eingeführt, werden Analytische Funktionen außerhalb des DWH-Umfeldes oft ignoriert oder einfach nur nicht beachtet. Dabei können diese Funktionen auch bei klassischen Auswertungen gute Dienste leisten. Der wesentliche Unterschied zwischen Analytischen

Funktionen und den Aggregatfunktionen ist hierbei, dass Analytische Funktionen mehr als eine Zeile zurückliefern können.

Die generelle Syntax für Analytische Funktionen lautet:

```
analytical_function
  ( param_1, …, param_n )
  over ( [partition by <…>]
         [order by <…>]
         [<window_clause>] )
```

Count

Selbst “alte Bekannte” wie die count-Funktion können damit komplexere Aufgaben lösen, als bisher gedacht. Abfragen wie „Wie viele Tunnel gibt es insgesamt?“ oder „Wie viele Tunnel gibt es pro Strecke?“ sind relativ trivial:

```
select count ( * ) as anzahl_tunnel
  from tunnel;

ANZAHL_TUNNEL
-------------
          685

select strecke_nr, count ( * ) as anzahl_tunnel
  from tunnel
 group by strecke_nr
 order by strecke_nr;

STRECKE_NR ANZAHL_TUNNEL
---------- -------------
      1239             4
      1270             7
      1271             4
      1713             1
      1733            63
…
```

Spannender ist in diesem Zusammenhang die Frage “wieviele Tunnel mit einem Abstand von +/-2km liegen um jeden Tunnel herum?“. Auch dies ist mit der count-Funktion zu ermitteln:

```
select bezeichnung as tunnel_name
     , count ( * ) over ( partition by strecke_nr order by von_km_r
                          range between 2 preceding and 2 following ) – 1
       as anzahl_nachbarn
  from tunnel
 where bezeichnung is not null
 order by 2 desc;

TUNNEL_NAME               ANZAHL_NACHBARN
------------------------- ---------------
Hohenacker-                             9
4.Bauer-                                8
3.Bauer-                                8
Farrenhalde-                            8
```

```
Steinbis-                              8
NordSüdFernb.                          8
NordSüdFernb.                          8
Tunnel Lenne-Dreick                    8
NordSüdFernb.                          8
NordSüdFernb.                          8
3.Glasträger-                          7
…
```

Da jedes Objekt auch sich selbst als Nachbar findet, wird vom Ergebnis 1 abgezogen, um die Anzahl der wirklichen Nachbarn zu ermitteln.

listagg

Eine neue Funktionalität in Oracle 11g ist die Funktion „listagg“. In ihrer einfachsten Verwendung können damit die Namen aller Tunnel in eine Zeile projiziert werden. Hierbei ist zu beachten, dass dies nur bis 4000 Zeichen Gesamtlänge möglich ist. Folgendes Beispiel fügt alle Tunnelnamen getrennt durch ein Semikolon zusammen:

```
select listagg ( bezeichnung, '; ' )
         within group ( order by strecke_nr, km_von_e ) tunnel_liste
  from tunnel
 where strecke_nr < 1500;

TUNNEL_LISTE
--------------------------------------------------------------------------------
U S-B Trogbauw. BA 3; U S-B Rechteckt. BA 4; F1; F2; U Tunnel Hbf-Jungfern;
U Tunnel Jungf-Stadth; U Tunnel Stadth-Landb; tlw.Bf L,Tübb; tlw.Bf L,Tübb;
U Tunnel Reep-Königst; U Tunnel Königstr-Alt; U Tunnel Ramp-Hp Harb; U Tun
Hp Harb-Bf Rath; U Tun Bf Rath-Hp Heim; U Tun Heimf-Tu.Mund
```

Ein wenig übersichtlicher gestaltet sich diese Liste bereits, wenn die Tunnelnamen pro Streckennummer aufgelistet werden:

```
select strecke_nr
     , listagg( bezeichnung, '; ' ) within group ( order by km_von_e ) tl
  from tunnel
  group by strecke_nr
  order by strecke_nr;

STRECKE_NR TUNNEL_LISTE
---------- ----------------------------------------------------
      1239 U S-B Trogbauw. BA 3; U S-B Rechteckt. BA 4; F1; F2
      1270 U Tunnel Hbf-Jungfern; U Tunnel Jungf-Stadth;
           U Tunnel Stadth-Landb; tlw.Bf L,Tübb; tlw.Bf L,Tübb;
           U Tunnel Reep-Königst; U Tunnel Königstr-Alt
      1271 U Tunnel Ramp-Hp Harb; U Tun Hp Harb-Bf Rath;
           U Tun Bf Rath-Hp Heim; U Tun Heimf-Tu.Mund
      1713 U Flughafenbahnhof
…
```

rank

Mit der Funktion rank können die Platzierungen von Daten an Hand eines Sortierkriteriums ermittelt werden. Um die Reihenfolge der Tunnel pro Strecke darzustellen, ist folgendes Statement geeignet:

```
select bezeichnung tunnel_name
     , strecke_nr
     , km_von
     , km_bis
     , rank () over ( partition by strecke_nr
                      order by km_von_e ) position_auf_strecke
  from tunnel;
```

```
TUNNEL_NAME     STRECKE_NR KM_VON     KM_BIS     POSITION_AUF_STRECKE
--------------- ---------- ---------- ---------- --------------------
U S-B Trogbauw.       1239 11,6 + 92  11,8 + 11                     1
 BA 3
U S-B Rechteckt       1239 11,8 + 11  11,9 + 77                     2
. BA 4
F1                    1239 12,0 + 2   13,7 + 50                     3
F2                    1239 12,0 + 2   13,7 + 50                     3
U Tunnel Hbf-Ju       1270 0,2 + 64   1,1 + 45                     1
ngfern
U Tunnel Jungf-       1270 1,4 + 28   1,8 + 39                     2
Stadth
U Tunnel Stadth       1270 2,1 + 33   2,8 + 83                     3
-Landb
tlw.Bf L,Tübb         1270 3,1 + 27   3,8 + 10                     4
tlw.Bf L,Tübb         1270 3,1 + 27   3,8 + 10                     4
U Tunnel Reep-K       1270 4,1 + 49   4,1 + 49                     6
önigst
U Tunnel Königs       1270 5,0 + 68   5,0 + 68                     7
tr-Alt
```

lead / lag

Mit Hilfe von lead und lag lassen sich Vorgänge- und Nachfolgedatensätze ermitteln. Für einige Auswertungen wird von jedem Tunnel auf einer Strecke der davor und der dahinterliegende benötigt.

```
select bezeichnung tunnel_name
     , strecke_nr
     , km_von
     , km_bis
     , lag ( bezeichnung ) over ( partition by strecke_nr
                                  order by km_von_e ) vorgaenger
     , lead ( bezeichnung ) over ( partition by strecke_nr
                                   order by km_von_e ) nachfolger
  from tunnel
order by strecke_nr
       , km_von_e;
```

```
TUNNEL_NAME   STRECKE_NR KM_VON     KM_BIS     VORGAENGER    NACHFOLGER
------------- ---------- ---------- ---------- ------------- -------------
U S-B Trogbau       1239 11,6 + 92  11,8 + 11                U S-B Rechtec
w. BA 3                                                      kt. BA 4
U S-B Rechtec       1239 11,8 + 11  11,9 + 77  U S-B Trogbau F2
kt. BA 4                                       w. BA 3
F2                  1239 12,0 + 2   13,7 + 50  U S-B Rechtec F1
                                               kt. BA 4
F1                  1239 12,0 + 2   13,7 + 50  F2
U Tunnel Hbf-       1270 0,2 + 64   1,1 + 45                 U Tunnel Jung
Jungfern                                                     f-Stadth
U Tunnel Jung       1270 1,4 + 28   1,8 + 39   U Tunnel Hbf- U Tunnel Stad
f-Stadth                                       Jungfern      th-Landb
U Tunnel Stad       1270 2,1 + 33   2,8 + 83   U Tunnel Jung tlw.Bf L,Tübb
th-Landb                                       f-Stadth
tlw.Bf L,Tübb       1270 3,1 + 27   3,8 + 10   U Tunnel Stad tlw.Bf L,Tübb
                                               th-Landb
tlw.Bf L,Tübb       1270 3,1 + 27   3,8 + 10   tlw.Bf L,Tübb U Tunnel Reep
                                                             -Königst
U Tunnel Reep       1270 4,1 + 49   4,1 + 49   tlw.Bf L,Tübb U Tunnel Köni
-Königst                                                     gstr-Alt
U Tunnel Köni       1270 5,0 + 68   5,0 + 68   U Tunnel Reep
gstr-Alt                                       -Königst
```

Die Funktionen lag und lead benötigen immer die Angabe, welche Informationen des Nachfolgers bzw. Vorgängers ausgegeben werden sollen. Die Übergabe erfolgt in Form eines Parameters. Es ist auch möglich einen Offset zu definieren, um z.B. den übernächsten Tunnel zu ermitteln. Dies geschieht durch Angabe eines weiteren Parameters. In obigem Beispiel erfolgt dies durch den Aufruf von:

```
select bezeichnung tunnel_name
     , strecke_nr
     , km_von
     , km_bis
     , lag ( bezeichnung, 2 ) over ( partition by strecke_nr
                                     order by km_von_e ) vorgaenger
     , lead ( bezeichnung, 2 ) over ( partition by strecke_nr
                                     order by km_von_e ) nachfolger
  from tunnel
order by strecke_nr
       , km_von_e;
```

Reguläre Ausdrücke

Alphanumerische Spalten werden in Oracle klassischerweise mit den Funktionen like, substr, instr, translate & replace durchgeführt. Auch komplexe Verschachtelungen dieser Funktionen sind möglich. Allerdings sind einige Abfragen nur sehr komplex oder mit reinen SQL-Mitteln gar nicht machbar. Natürlich können PL/SQL-Funktionen erstellt werden, um die Aufgabe zu lösen. Reguläre Ausdrücke bieten die Möglichkeit, komplexe Operationen mit reinen SQL-Operationen durchzuführen.
In Eingangs erwähnter Infrastrukturdatenbank werden aus über 20 Quellsystemen Daten geladen. Natürlich hat jedes Quellsystem seine eigene Art und Weise, die KM-Werte der Lageangaben darzustellen. Je nach Quellsystem muss die Umwandlungsroutine, die zwischen den verschiedenen Schreibweisen umrechnet, ein klein wenig anders arbeiten. Aus diesem Grund wurde eine Funktion

entwickelt, die inzwischen über 20 verschieden Schreibweisen von Kilometrierungen versteht. Mit normalen replace und substr-Funktionalitäten wäre dies nicht zu schaffen gewesen.
Seit Oracle 10g gibt es die Möglichkeit, reguläre Ausdrücke in der Datenbank zu nutzen. Bisher gab es REGEXP_INSTR, REGEXP_SUBSTR und REGEXP_REPLACE. Ergänzt wird dies seit Oracle 11g Rel. 1 durch REGEXP_COUNT. Die Namenswahl erfolgte erfreulicherweise analog zu den bestehenden klassischen SQL-Funktionen INSTR, SUBSTR und REPLACE, so dass an Hand des Namens bereits zu erkennen ist, was die Funktionen tun.

Um alle Tunnel zu finden, deren Namen mit mindestens 10 Buchstaben (inkl. Leerzeichen) beginnt und am Ende beliebig viele Zahlen hat, genügt folgendes Statement:

```
select bezeichnung as tunnel_name
  from tunnel
 where regexp_instr ( bezeichnung, '^[a-zA-Z\ ]{10,}[0-9]{1,}$' ) = 1;

TUNNEL_NAME
-------------------------
Rauenthaler 2
Rauenthaler 1
Weyersberger1
Saffenburger2
Hinterburden2
Merchweiler 1
…
```

REGEXP_INSTR gibt die Position der Zeichenkette zurück, REGEXP_COUNT dagegen gibt an, wie oft eine bestimmte Zeichenkette innerhalb eines Textfeldes zu finden ist. Um rauszufinden, wie oft in einem Tunnelnamen 2-stellige numerische Angaben vorhanden sind, genügt folgendes Statement:

```
select bezeichnung as tunnel_name
     , regexp_count ( bezeichnung, '[0-9]{2}' ) as anzahl
  from tunnel
 order by 2 desc nulls last;


TUNNEL_NAME           ANZAHL
--------------------- ------
jetzt Strecke 3682         2
VP2 U Gl3600               2
S-B Bw44/45 BOS-SAL        2
jetzt Strecke 3682         2
U 4010 Krbw Zeppelinh      2
U A66                      1
U 509S Isart.-Rosenhp      1
U L 41.2                   1
…
```

Abfragen mit LIKE, SUBSTR, REPLACE sind, sofern die Abfrage damit realisiert werden kann, deutlich schneller als reguläre Ausdrücke. Dafür sind letztere deutlich mächtiger.
Oracle unterstützt nicht alle Möglichkeiten der regulären Ausdrücke. Eine Liste der unterstützten Funktionen findet sich in der „Oracle Database SQL Language Reference“ im Appendix „Oracle Regular Expression Support“.

Virtuelle Spalten (virtual columns)

Wie bereits oben erwähnt, verfügen alle Infrastrukturdaten über Lageinformationen in insgesamt drei verschiedenen Schreibweisen. Es gibt eine sog. Vermessungstechnische Schreibweise, eine Darstellung als Kommazahl und eine ganzzahlige Darstellung. Zwischen diesen Darstellungen gibt es Umrechnungsfunktionen. Beim Laden der Daten werden ausgehend von der vermessungstechnischen Schreibweise die anderen beiden berechnet und in entsprechende Spalten geschrieben. Dazu muss der Ladeprozess entsprechend erweitert werden.
Seit Oracle 11g gibt es die Möglichkeit, solche Informationen in Virtuelle Spalten auszulagern. Dazu wird bei der Spaltendefinition die Berechnungsvorschrift angegeben. Um die oben erwähnte Tabelle Tunnel mit zwei virtuellen Spalten zu versehen, die die Lageinformation als Kommazahl beinhalten, genügt folgendes DDL-Statement:

```
alter table tunnel
  add ( virtual_von_km_r as ( pck_util.get_real_km ( km_von_e ) )
      , virtual_bis_km_r as ( pck_util.get_real_km ( km_bis_e ) ) );
```

Zu beachten ist hierbei, dass der Alias-Name nicht wie gewohnt rechts vom Schlüsselwort AS steht sondern links.
Virtuelle Spalten können nicht als Basis weiterer virtueller Spalten genutzt werden (ORA-54012: virtual column is referenced in a column expression):

```
alter table tunnel
  add ( virtual_von_km_d as ( pck_util.get_db_km ( virtual_von_km_r ) )
      , virtual_bis_km_d as ( pck_util.get_db_km ( virtual_bis_km_r ) ) );
alter table tunnel
*
ERROR at line 1:
ORA-54012: virtual column is referenced in a column expression
```

Da sich per Definition virtuelle Spalten nur aus anderen Spalten derselben Tabelle errechnen, sind DML-Operationen nicht erlaubt:

```
update tunnel
   set  virtual_von_km_r = 2;
update tunnel
       *
ERROR at line 1:
ORA-54017: UPDATE operation disallowed on virtual columns


insert into tunnel ( bezeichnung, virtual_von_km_r )
  values ( 'Neuer Tunnel', 23 );
insert into tunnel ( bezeichnung, virtual_von_km_r )
            *
ERROR at line 1:
ORA-54013: INSERT operation disallowed on virtual columns
```

Virtuelle Spalten verhalten sich, von obigen Restriktionen abgesehen, gegenüber dem Anwender wie reguläre Spalten. Ob Spalten virtuell angelegt wurden, kann mit Hilfe des Data Dictionary-Views user_tab_cols herausgefunden werden:

```
select table_name, column_name, data_type
  from user_tab_cols
 where virtual_column = 'YES'

TABLE_NAME  COLUMN_NAME       DATA_TYPE
----------- ----------------- ----------
TUNNEL      VIRTUAL_VON_KM_R  NUMBER
TUNNEL      VIRTUAL_BIS_KM_R  NUMBER
```

Virtuelle Spalten können indiziert werden, um die Zugriffe zu beschleunigen. Ein solcher Index verhält sich wie eine function based Index.

Virtuelle Spalten helfen, Speicherplatz zu sparen und möglicherweise dabei, Spalten bereitzustellen, ohne vorhandene Ladeprozesse anpassen zu müssen. Da virtuelle Spalten aber erst beim Zugriff berechnet werden, ist diese Technik bei Abfragen, die hochperformant laufen sollen, intensiv zu testen.

With-Klausel (with clause)

Eingeführt in Oracle 9i Rel. 2 bietet die SQL-99-konforme With-Klausel die Möglichkeit, Select-Abfragen übersichtlich zu gestalten. Dies ist vor allem dann interessant, wenn mehrere Untermengen aus mehreren Tabellen gejoint werden sollen.
Um alle Tunnel der Strecke 3600 auszugeben genügt folgendes Statement:

```
with tunnel_der_strecke_3600
  as ( select bezeichnung tunnel_name
            , strecke_nr
            , km_von
            , km_bis
            , ra_laenge laenge
         from tunnel
        where strecke_nr = 3600 )
select *
  from tunnel_der_strecke_3600;

TUNNEL_NAME     STRECKE_NR KM_VON             KM_BIS                 LAENGE
--------------- ---------- ------------------ ------------------ ----------
Burghauner            3600 132,4 + 41         132,6 + 79                238
Bebenroth             3600 222,4 + 99         223,4 + 34                935
Cornberger            3600 177,8 + 14         178,5 + 34                719
Schürzeberg           3600 218,5 + 26         218,6 + 100               173
Schlüchterner         3600 78,1 + 86          81,7 + 60                3576
```

Die Aliasnamen der Sub-Selects können in weiteren With-Klauseln verwendet werden. Um z.B. aus obiger Liste alle Tunnel zu ermitteln, die eine Länge von mehr als 500m haben, ist folgendes Statement geeignet:

```
with tunnel_der_strecke_3600
  as ( select bezeichnung tunnel_name
            , strecke_nr
            , km_von
            , km_bis
            , ra_laenge laenge
         from tunnel
        where strecke_nr = 3600 )
, tu_der_str_3600_laenger_500
  as ( select tunnel_name
            , strecke_nr
            , km_von
            , km_bis
            , laenge
  from tunnel_der_strecke_3600
 where laenge >= 500 )
select *
  from tu_der_str_3600_laenger_500;
```

```
TUNNEL_NAME     STRECKE_NR KM_VON             KM_BIS                LAENGE
--------------- ---------- ------------------ ------------------ ----------
Bebenroth             3600 222,4 + 99         223,4 + 34                935
Cornberger            3600 177,8 + 14         178,5 + 34                719
Schlüchterner         3600 78,1 + 86          81,7 + 60                3576
```

Im genannten Beispiel ist die Verwendung der with clause vielleicht ein wenig übertrieben, aber im Alltag finden sich viele Anwendungsfälle, die durch Verwendung der with clause an Übersichtlichkeit gewinnen. Nutzer von graphischen Werkzeugen, wie z.B. Toad profitieren darüber hinaus von der Möglichkeit, Teile des Select-Statements auszublenden, um sich z.B. bei der Fehlersuche auf den interessanten Teil konzentrieren zu können (siehe Abbildung 1).

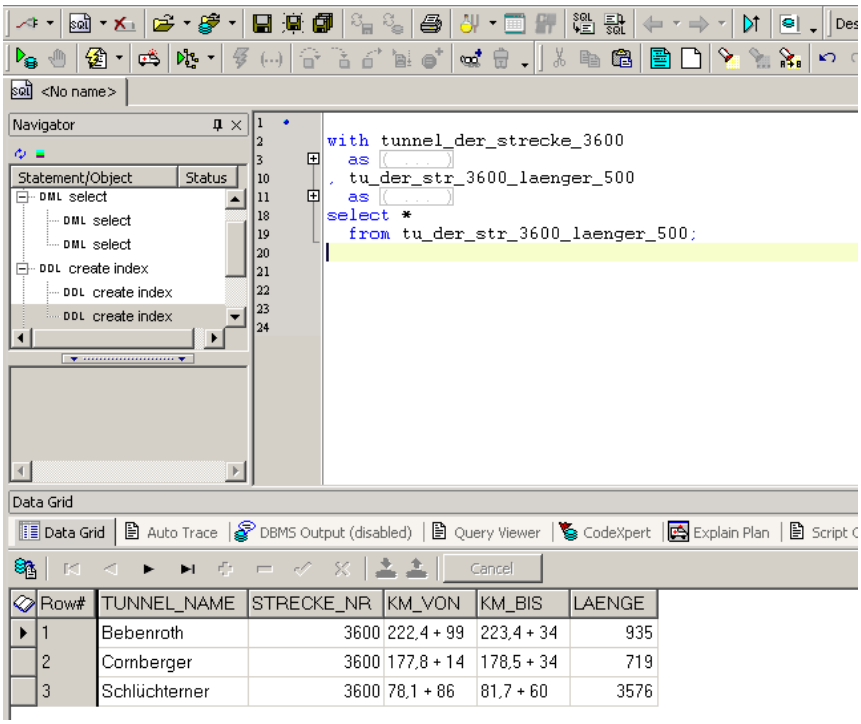

Abbildung 1: Nutzung der with clause in graphischen Werkzeugen

Zusammenfassung

Die genannten Funktionalitäten haben sich im Rahmen der Infrastrukturdatenbank und darüber hinaus auch in anderen Projekten als äußerst hilfreich erwiesen. Einige neue Funktionen scheinen auf dem Weg, sich in diese Liste einzureihen. Der Vortrag erhebt keinen Anspruch auf Vollständigkeit und natürlich ist es jedem selbst überlassen, welche der div. Möglichkeiten einer Oracle-Datenbank genutzt werden. Dieser Vortrag soll als Anregung verstanden werden, sich mit neuen oder auch bereits seit einiger Zeit vorhandenen Funktionalitäten auseinanderzusetzen – für einige wird es Verwendungszwecke geben und vielleicht wird einiges einfacher.

Kontaktadresse:

Jens Behring
its-people Hochtaunus GmbH
Lyoner-Straße 44-48
D-60528 Frankfurt

Telefon:	+49 (0) 69-24752100
Fax:	+49 (0) 69-247521021
E-Mail	jens.behring@its-people.de
Internet:	www.its-people.de