Abstimmen mit dem Gupta Report Builder

Ergebnisse von Abstimmungen werden im Internet häufig als Punktreihen dargestellt. Aber ist es auch möglich eine solche einfache Darstellung einer Abstimmung mit dem Report Builder (RB) nachzubilden? In der Tat, »der alte Bastard von einem Report Builder« liefert auch hier eine schnelle Lösung.

von Thomas Wiedmann

Im Internet finden sich zahlreiche Abstimmungen, deren Ergebnis in waagerechten Punktreihen dargestellt wird, dies ist eine einfache aber wirkungsvolle Darstellungsform. Im folgenden Artikel sehen Sie, was zu tun ist, um den RB zu überreden, eine ähnliche Darstellung zu erzeugen (Bild 1).

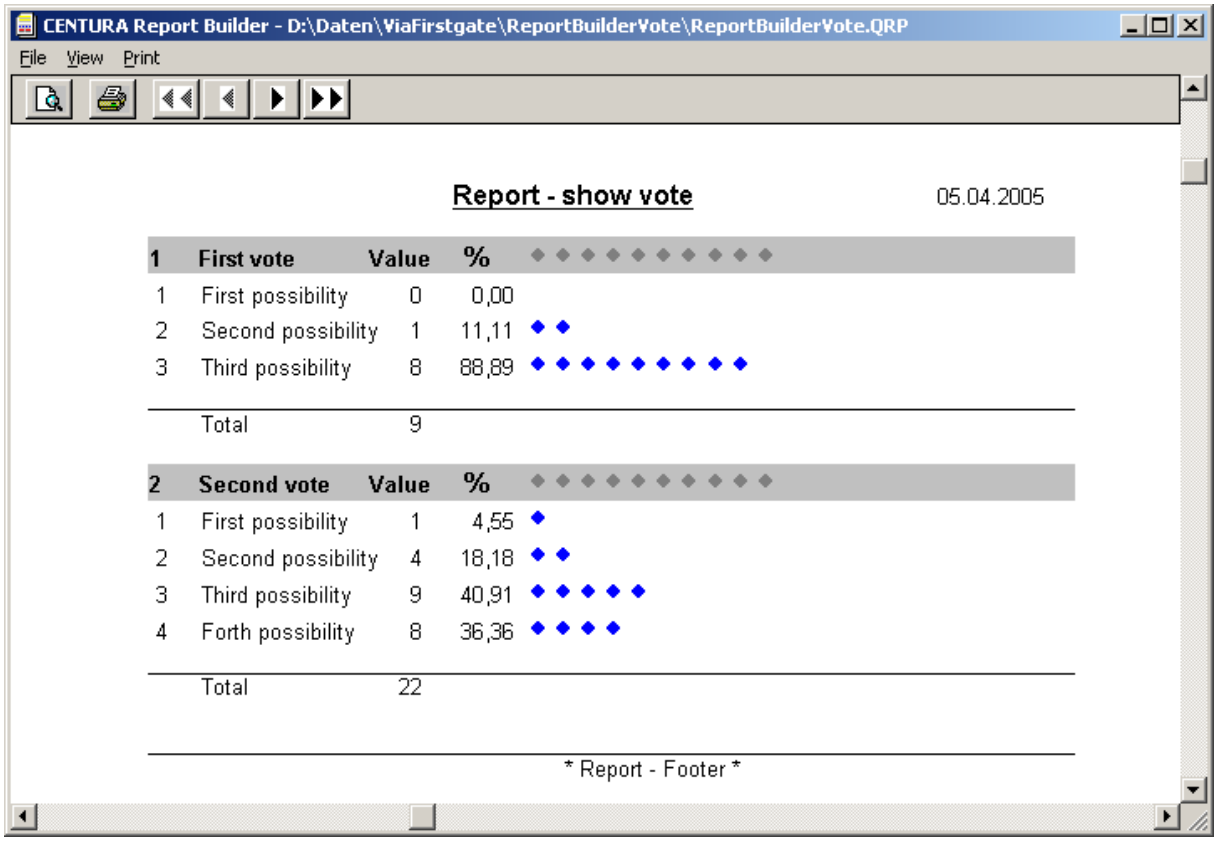


Bild 1: Zeigt die gewünschte Darstellung einer Abstimmung mit Hilfe des RB

Das Bild 1 zeigt die Ergebnisse zweier Abstimmungen (First vote und Second Vote). In der ersten Abstimmung gab es drei, in der zweiten Abstimmung vier Auswahlmöglichkeiten. Das Abstimmungsergebnis wird als absolute Zahl, als Prozentanteil und als Punktreihe dargestellt. Wie es zu dieser Punktreihe kommt, ist das zentrale Thema dieses Artikels. Folgende Komponenten werden im Artikel eingesetzt:

- Windows 2000
- Gupta Team Developer v2.1 PTF2 [1]
- Gupta Report Builder v2.1 PTF2
- SQLBase v7.5

Doch bevor es losgeht, zeige ich Ihnen die zugrunde liegende Datenstruktur (SQLBase).

Datenbasis

Grundlage ist die Mastertabelle *VOTE_TYPE*, in der die beiden Abstimmungen angelegt sind.

```
CREATE TABLE vote_type (
 type_id INT NOT NULL,
 type_name VARCHAR(50) NOT NULL,

 PRIMARY KEY (type_id)
);
```

Die einzelnen Abstimmungsergebnisse sind in der Detailtabelle VOTE_VALUE gespeichert.

```
CREATE TABLE vote_value (
 value_id INT NOT NULL,
 type_id INT NOT NULL,
 value INT NOT NULL,
 value_order INT NOT NULL,
 value_name VARCHAR(50) NOT NULL,

 PRIMARY KEY (value_id),

 FOREIGN KEY fk_vote_value (type_id)
  REFERENCES vote_type
  ON DELETE CASCADE
);
```

Das vollständige SQL-Script (CREATE.DDL), um die Tabellen und die notwendigen Indices zu erzeugen, liegt diesem Artikel natürlich bei. Nachdem die Tabellen existieren, können die Testdaten eingelesen werden. Diese habe ich im SQL-Script (INSERT.SQL) zusammengefasst. Nachdem die Tabellen gefüllt sind, geht es nun um die Kommunikation zwischen dem Gupta Programm und der RB-Engine.

Client – Server communication ?

Ähnlich einer Client/Server Verbindung arbeiten der Gupta Team Developer (TD) und der RB zusammen, allerdings wechseln sie mitunter die Seiten. Nachdem der TD mit *SalReportView()* den RB gestartet hat, fordert der RB mit Hilfe seiner speziellen *SAM_Report** Messages die Daten vom TD an.

```
ReportBuilderVote.apt:

Function: fPrintReport()

[...]
If p_bReportView = TRUE
 If SalReportView(hWndForm,hWndNULL,lsReportFilename,lsInput,lsVar,lnFlag)
  !
 Else
  Call SalMessageBox('SalReportView failed...','ERROR',MB_IconStop)
Else
 If SalReportPrint(hWndForm,lsReportFilename,lsInput,lsVar,0,RPT_PrintAll,
    0,0,lnFlag)
  !
 Else
  Call SalMessageBox('SalReportPrint failed...','ERROR',MB_IconStop)
[...]
```

Wenn *SalReportView()* oder *SalReportPrint()* erfolgreich ausgeführt worden ist, sendet der RB spezielle *SAM_Report** Messages, die auf ganz bestimmte Art und Weise beantwortet werden müssen. Auf Form-Window-Level müssen diese Messages geprüft und beantwortet werden, da wir dem Report beim Start das Fenster-Handle in Form von *hWndForm* übergeben haben.

```
Message Actions
```

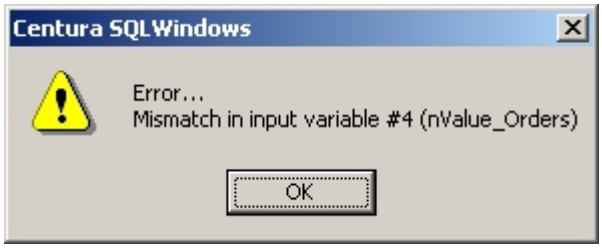
```
! /* handle report messages
On SAM_ReportStart
On SAM_ReportFetchInit
On SAM_ReportFetchNext
On SAM_ReportNotify
On SAM_ReportFinish
```

Die Beantwortung der einzelnen Report Messages habe ich jeweils zu einer Funktion zusammengefasst.

```
Functions
[...]
! /* privat report functions
Function: _OnReportStart
Function: _OnReportFetchInit
Function: _OnReportFetchNext
Function: _OnReportNotify
Function: _OnReportFinish
[...]
```

SAM_ReportStart

Zuerst sendet der RB die Nachricht *SAM_ReportStart,* dies ist der Zeitpunkt, um den SQL Befehl auszuführen, siehe *_OnReportStart()*. Die Nachricht *SAM_ReportStart* wird nur einmal gesendet. Das Ergebnis der SQL Abfrage wird später (auf Anforderung) dem Reportbuilder Zeile für Zeile mit Hilfe der Report *Input Items* übergeben. Absolut wichtig ist hier die identische Reihenfolge und Schreibweise der Variablen im TD als auch den *Input Items* des RB. Sollte hier auch nur ein kleiner Unterschied sein, verweigert der RB den Dienst (Bild 2).



Pic. 2: Mismatch between TD vars and RB Input Items

Der Fehler »Mismatch in input variable #4« weist auf die vierte Variable *nValue_Orders*. Hier liegt ein Unterschied in der Schreibweise zwischen den definierten Variablen im APT und den *Report Input Items* vor. Korrekterweise muss es *nValue_Order* lauten. Ist dieser Fehler korrigert, sendet der RB als nächstes die Nachricht *SAM_ReportFetchInit.*

SAM_ReportFetchInit

Hiermit kündigt der RB an, dass er nun die erste Row (aus dem Resultset der Abfrage) anfordern wird. Hier initialisiere ich *bFetchFirst* auf TRUE. Bitte beachten Sie, dass die Nachricht *SAM_ReportFetchInit* unter Umständen zweimal gesendet wird und zwar immer dann, wenn nach *SalReportView()* der angezeigte Report doch noch auf Papier gedruckt wird. Ob der SQL-Befehl durch *SAM_ReportStart* oder *SAM_ReportFetchInit* ausgelöst wird, hat unterschiedliche Konsequenzen. Wenn Sie maximale Datenkonsistenz zwischen View und Report möchten, dann ist *SAM_ReportStart* und der Isolationlevel *Read Repeatability* (RR) die erste Wahl, falls Sie eine optimale concurrency benötigen, ist *SAM_*ReportFetchInit ergänzt um den Isolationlevel RS oder CS zu empfehlen. Sie sehen, diese Thematik ist komplexer als man im ersten Moment denkt. Für welchen Lösungsweg Sie sich entscheiden,

muss von Fall zu Fall neu überdacht werden. Das Problem ist letztlich nicht abschließend lösbar.

SAM_ReportFetchNext

Der RB ist nun bereit und fordert die erste beziehungsweise die nächste Row an. Im folgenden Codeteil möchte ich nochmals mein Konzept auf Basis der Variablen *bFetchFirst* aufzeigen.

```
[...]
If bFetchFirst = TRUE
 Set bFetchFirst = FALSE
  If SqlFetchRow(hSqlA,0,lnFetchResult)
   If lnFetchResult = FETCH_Ok
    Return TRUE
   Else
    Return FALSE
  Else
   Return FALSE
Else
 If SqlFetchNext(hSqlA,lnFetchResult)
  If lnFetchResult = FETCH_Ok
   Return TRUE
  Else
   Return FALSE
 Else
  Return FALSE
[...]
```

Nach einem *SAM_ReportFetchInit* wird in *SAM_ReportFetchNext* einmalig ein *SqlFetchRow (hSqlA,0,lnFetchResult)* durchgeführt. Alle weiteren *SAM_ReportFetchNext* lösen *SqlFetchNext(..)* aus.

SAM_ReportNotify

Diese Report Message bietet dem Programmierer die Möglichkeit, direkt in den Reportablauf einzugreifen und zusätzliche Aktionen wie weitere SQL-Queries oder Report Variablen vom Typ Input Variables zu setzen. Es gibt eine ganze Reihe von diesen *Notify*-Nachrichten, beispielsweise *RPT_BeforeDetail.* Hierzu lohnt sich ein Blick in die Dokumentation zu *SAM_ReportNotify*.

```
Function: _OnReportNotify
 [...]
 Parameters
  Number: p_wParam
  Number: p_lParam
  [...]
 Actions
  Set lhWndReport = SalNumberToWindowHandle(p_wParam)
  !
  Select Case (p_lParam)
   Case RPT_BeforeDetail
    !
    ! /* Calc 100% and split into 10 pieces
    Set lnValueSum = _fQueryValueSum(nType_Id)
    Set lnStep = lnValueSum / 10
    Call SalReportSetNumberVar (lhWndReport, 'nProzent', (100/lnValueSum*nValue) )
    !
    ! /* Set Report Input Variables. SetNumberVar FALSE or TRUE
    Call SalReportSetNumberVar (lhWndReport, 'bShowPoint01',
         _fCheck(nValue,lnStep*0) )
    Call SalReportSetNumberVar (lhWndReport, 'bShowPoint02',
        _fCheck(nValue,lnStep*1) )
    [...]
    Break
```

Den notwendigen Report Handle *lhWndReport* läßt sich mit hilfe des ersten Parameters *p_wParam* und der Funktion *SalNumberToWindowHandle()* ermitteln. Der zweite Parameter *p_lParam* liefert den notification Wert, im obigen Beispiel die Reportkonstante *RPT_BeforeDetail*. Diesen Wert sendet der RB, bevor er daran geht, den Detailblock des QRP zu füllen. Mit Hilfe von *SalReportSetNumberVar()* können nun die speziellen Report *Input Variables* im QRP gesetzt werden. Für das weitere Verständnis ist nun ein Blick auf den Report (QRP) notwendig (Bild 3).

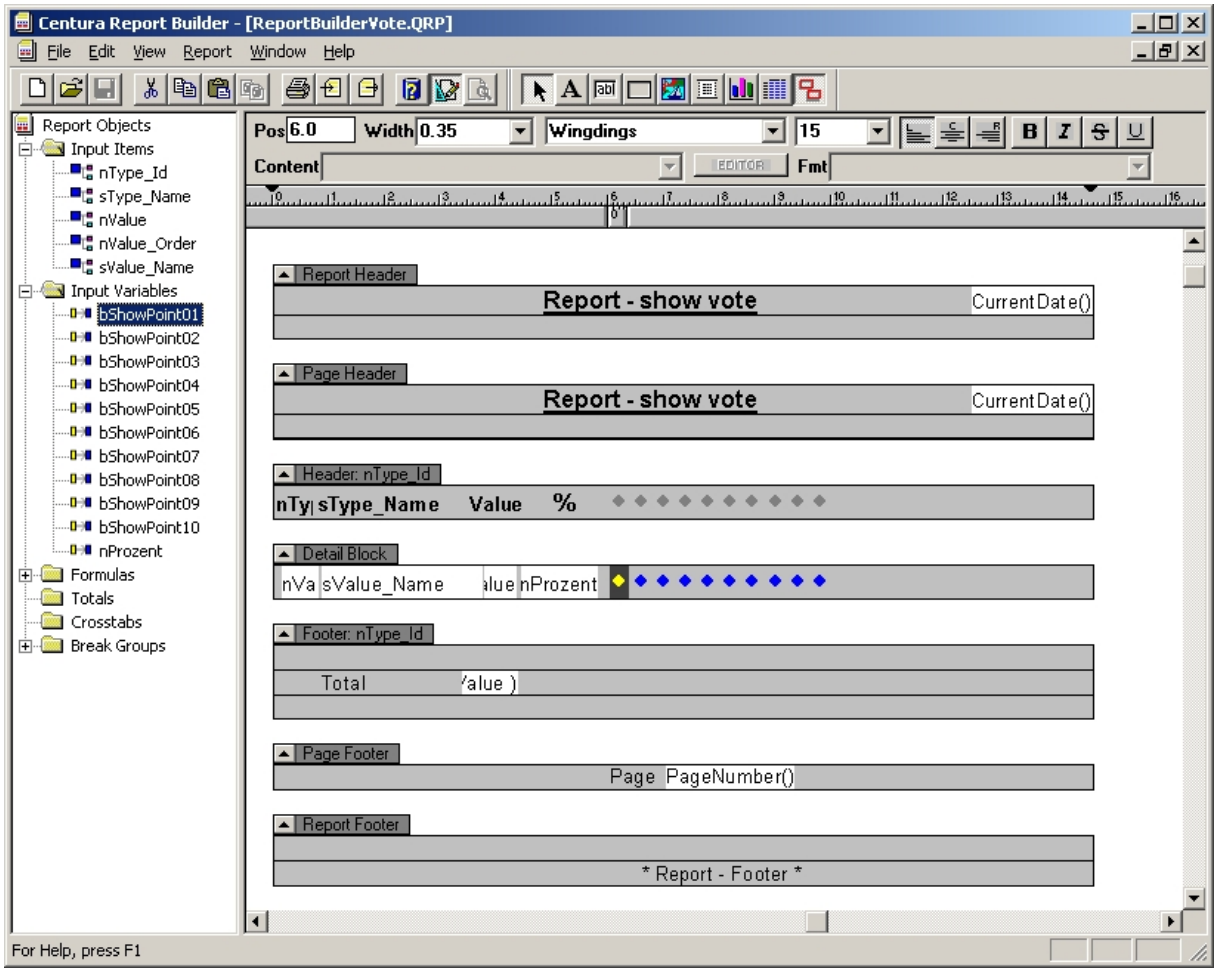


Bild 3: Report im Design Modus

Die Input Variable *bShowPoint01* ist dafür zuständig, ob die erste Raute angezeigt wird oder nicht. Hierzu wird per *SalReportSetNumberVar()* TRUE oder FALSE übergeben. Welcher BOOLEAN Wert übergeben wird, ermittelt die Function *_fCheck()*. Bleibt noch die Frage, woher das QRP weiß, ob die Raute angezeigt werden soll oder nicht. Das hierfür zuständige Property ist über das Kontextmenu der Raute erreichbar (Bild 4).

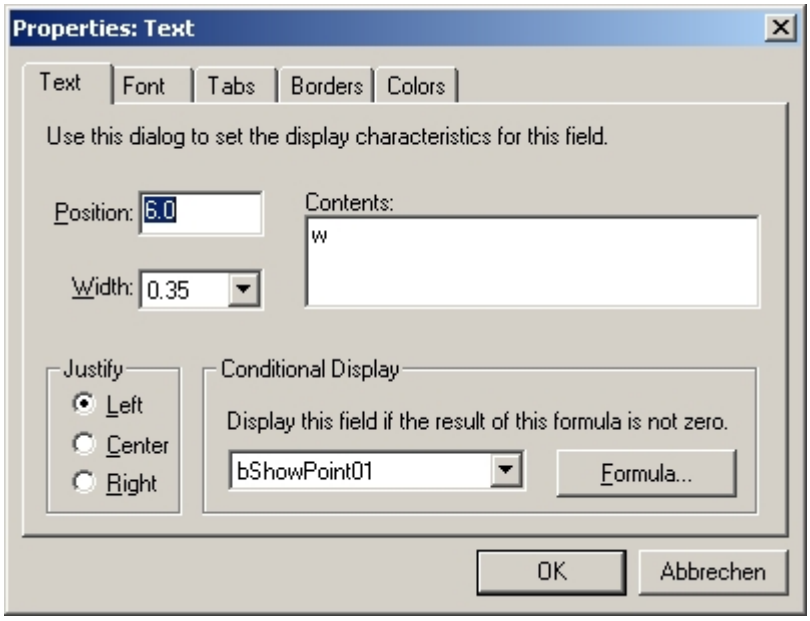


Bild 4: *Properties..* Kontextmenu jeder Raute

Im Contextmenu jeder Raute ist unter *Display this field if..* eine der zehn boolean Variablen *bShowPoint01* bis *..10* zugewiesen. Der RB bietet die Möglichkeit, einzelne Objekte abhängig von dem Wert einer *Input Variable* anzuzeigen oder auszublenden. Dies ist der eigentliche Kniff. Nebenbei sehen Sie noch, was notwendig ist, um eine *Gupta-typische* Raute zu erzeugen: Ein kleines w und die Schriftart *Wingdings.* Jede Raute ist somit nur ein simpler Background Text.

Finally

Der *Gupta Report Builder* kennt eine Menge kleiner Tricks, um spezielle Aufgaben zu lösen. Für komplexe Reports werden zwar häufig und zurecht andere Reportgeneratoren empfohlen, aber diese anderen Programme müssen zusätzlich lizensiert und erlernt werden. Der RB ist ein Teil des *Gupta Team Developer* und somit schon bezahlt.

Thomas Wiedmann ist Anwendungs- und Datenbankentwickler für individuelle Softwarelösungen, Autor des Buches »DB2«, sowie IBM Certified DB2 Administrator. Wenn er nicht gerade Musik hört, liest oder fotografiert, beschäftigt er sich mit Datenbanken und Web-Applikationen.

Literatur und Links

[1] Gupta: http://www.guptaworldwide.com/