Handbuch

DIMOWA@ SQL Resource Creator

Version 1.4

Autor des Dokuments	Häntschel	Erstellt am	01.01.2017
Seitenanzahl	15	DIMOWA®	

Historie der Dokumentversionen

Version	Datum	Autor	Änderungsgrund / Bemerkungen
0.1	01.01.17	Häntschel	Ersterstellung
0.2	22.02.17	Häntschel	Shortcuts hinzugefügt
0.3	21.04.17	Häntschel	Screenshots, Shortcuts
0.4	27.03.17	Häntschel	Screenshots, Update
0.5	07.10.17	Häntschel	Verschlüsselung

Inhaltsverzeichnis

Historie der Dokumentversionen....2
Inhaltsverzeichnis....2
1 Einleitung....3
1.1 Allgemeines....3
2 Programm....3
2.1 Allgemein....3
2.2 Vorgabe: Ressource....3
2.3 Vorgabe: Definieren der Ablage der Statements....3
2.4 Allgemeine Einstellungen....4
2.5 SQL Farben....5
2.6 Definieren Datenbanktypen....5
2.7 Definieren Projekt....6
2.8 Update....7
2.9 Status der SQL....7
2.10 Header im SQL....8
2.11 GUI....8
2.12 Shortcuts....9
3 Codebeispiele Delphi....10
3.1 Aufnehmen der Ressource....10
3.2 Konstante für Datenbanken (einfacher)....10
3.3 Konstante für Key....10
3.4 SQL aus Ressource extrahieren (Common) _Klartext....10
3.5 SQL aus Ressource extrahieren (Common) _Verschlüsselung....10
3.6 TTools_Verschlüsselung (DEC 5.2)....11
3.7 SQL aus Ressource extrahieren ohne Parameter....12
3.8 SQL aus Ressource extrahieren mit Parameter....12
3.9 SQL aus Ressource extrahieren mit Parameter und Platzhaltern....13
3.10 Compilieren....13
3.11 fertige Ressource (Klartext)....14
3.12 fertige Ressource (verschlüsselt)....14
4 Anhang / Ressourcen....15

1 Einleitung

1.1 Allgemeines

Der SQL Resource Creator verwaltet die SQL Statements verschiedener Datenbanken. Das Programm erstellt die Ressourcen Datei für das Einbinden in verschiede Programmiersprachen. Dieses Handbuch beschreibt die Verwendung des Programmes.

2 Programm

2.1 Allgemein

- Hinzufügen
- Kopieren
- Entfernen
- Bearbeiten

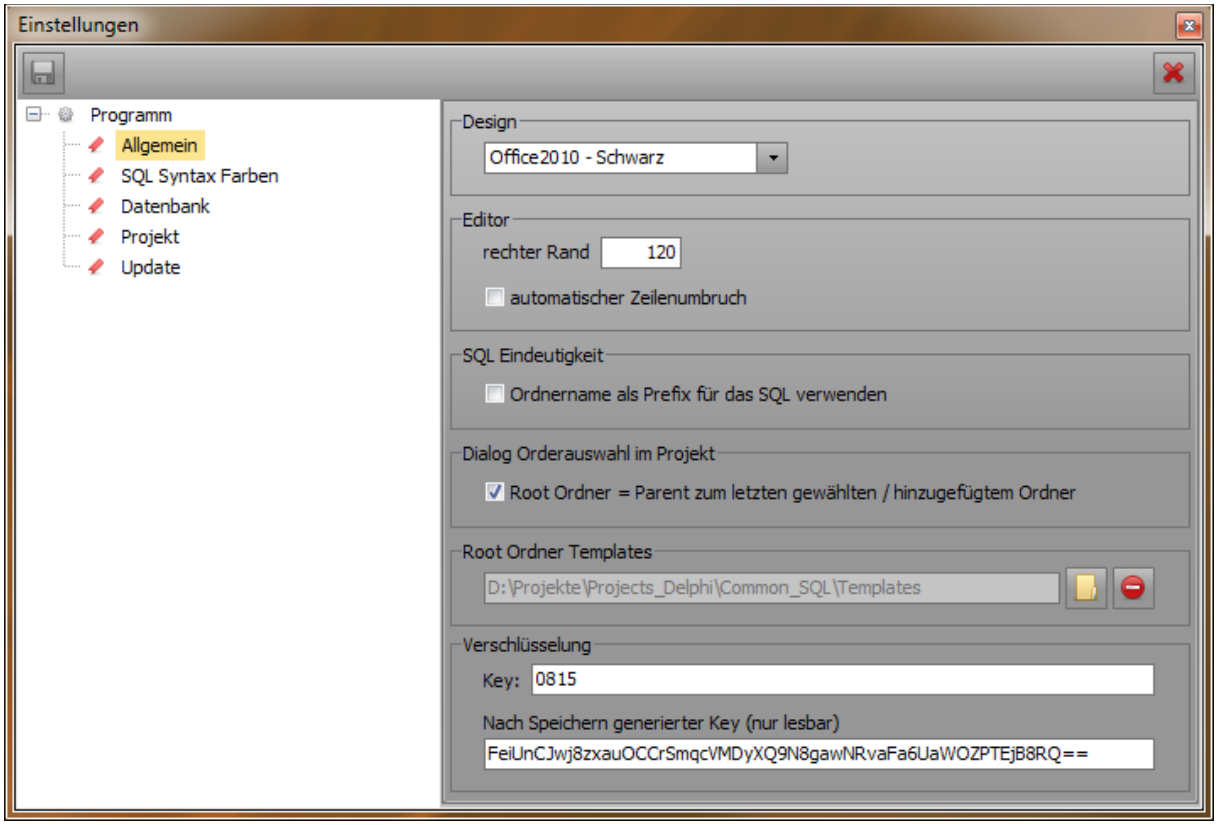
2.2 Vorgabe: Ressource

- Definieren der Ressource als Textdatei. (****.rc)
- ggf. ins Projekt aufnehmen

2.3 Vorgabe: Definieren der Ablage der Statements

- Gemeinsamer Order (beliebig) für SQL Statements (ggf. SVN)

2.4 Allgemeine Einstellungen



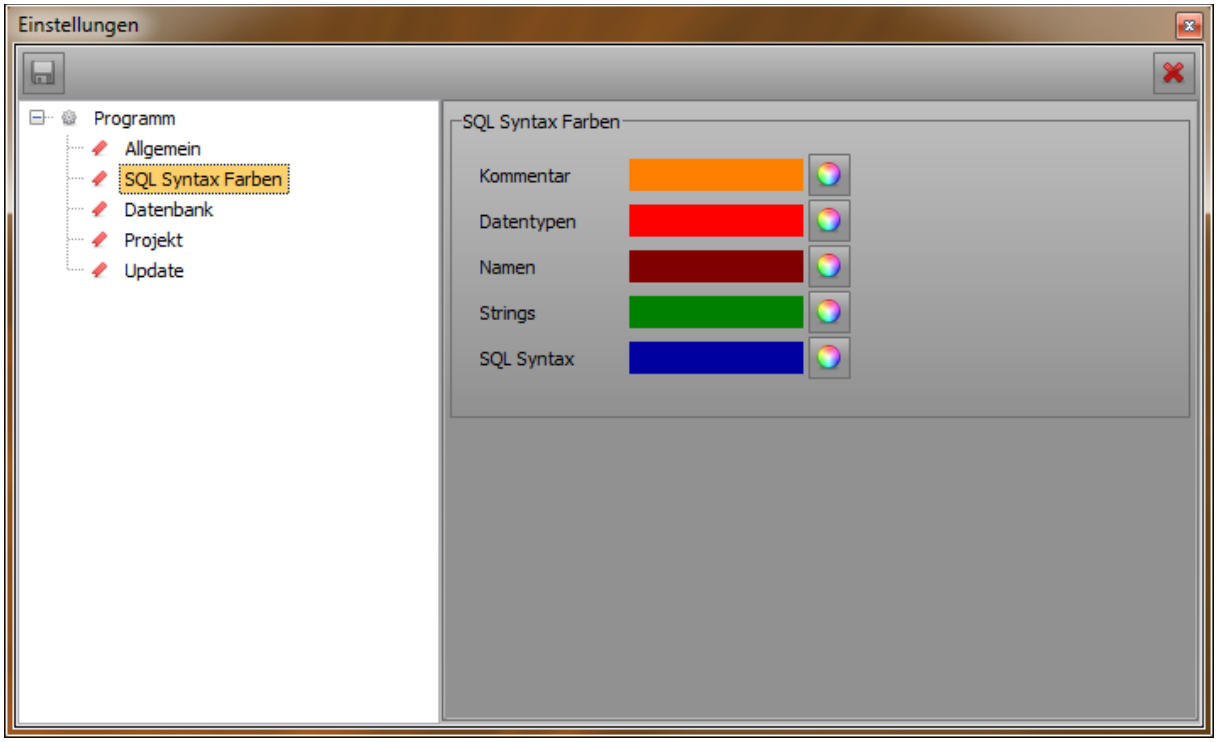
Editor: Der rechte Rand markiert die Anzahl der Zeichen für den Druck. Mit dem automatischem Zeilenumbruch wird das SQL Statement am Rand des Editors umgebrochen.

SQL Eindeutigkeit: Wenn markiert, wird für den Ressourcennamen der Name aus dem Ordner und dem SQL Namen zusammengesetzt. Das ist hilfreich wenn SQL Statements in den Ordnern die gleichen Namen tragen. Innerhalb eines Ordners sind keine gleichen Namen erlaubt.

Verschlüsselung: Vorgabe Schlüssel für die Speicherung der SQL Statements. Dieser Key wird intern mit einem Master Key verschlüsselt und in die INI gespeichert.

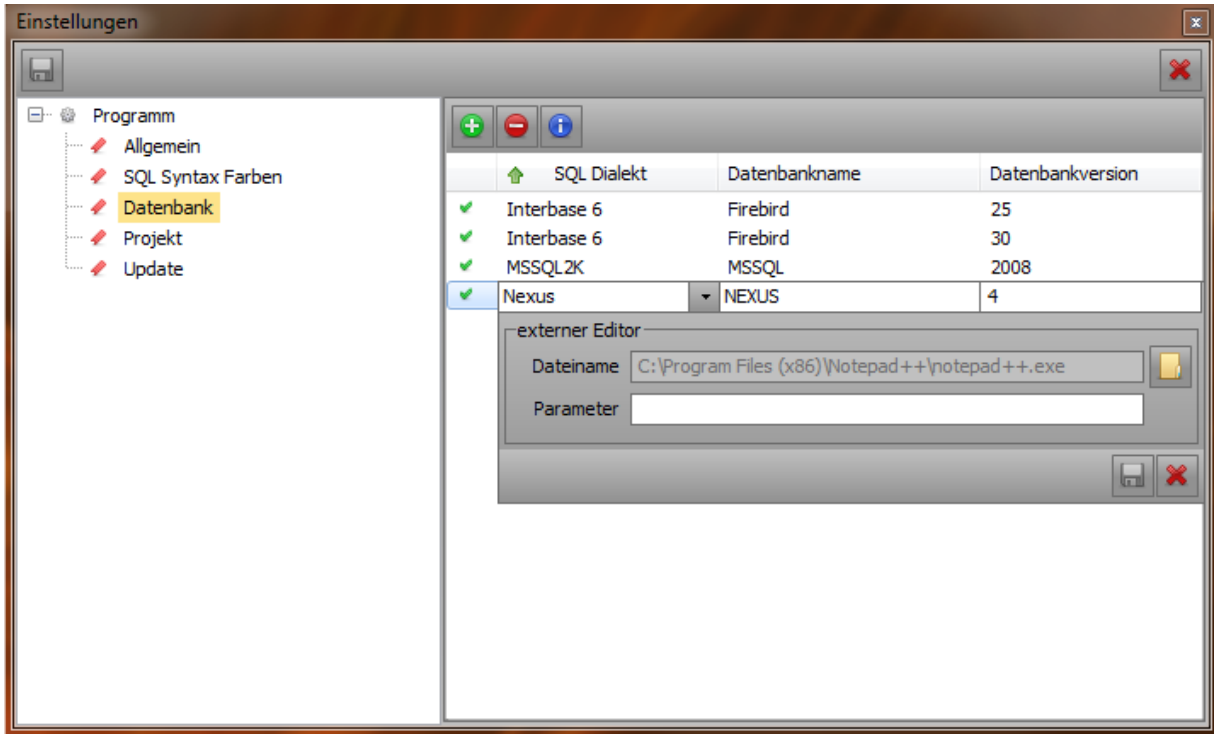
Ist das Projekt verschlüsselt, wird in der Ressource beim Datenbanktyp der Suffix „_Encrypted“ angehangen.

2.5 SQL Farben



Farben: Farben für den SQL Dialekt.

2.6 Definieren Datenbanktypen

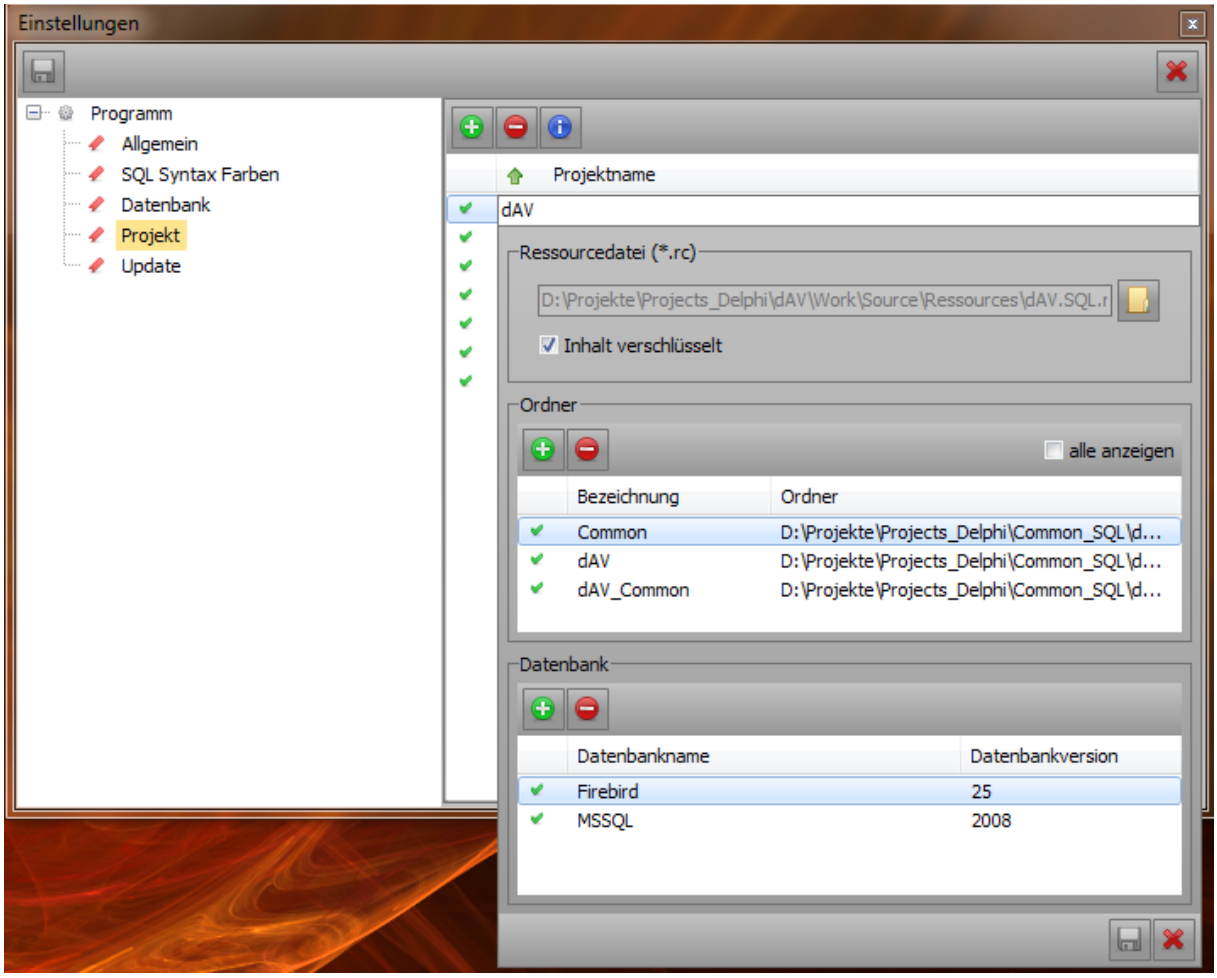


SQL Dialekt: Der Dialekt hat auf den Editor Auswirkung. Reserviere Wörter werden hervorgehoben. Standard ist für alle anderen die nicht gelistet sind.

externer Editor: Pfad, Name und Parameter (wie in der Konsole **ohne** die SQL Datei) für den Editor

Hinweis: Wenn Datenbanken geändert /gelöscht werden müssen sie neu im Projekt zugeordnet werden.

2.7 Definieren Projekt



Verschlüsselung: Ist die Option gewählt dann wird die verschlüsselte Datei in die Ressource aufgenommen. Damit erscheinen die SQL Statements nicht mehr im Klartext in der EXE.

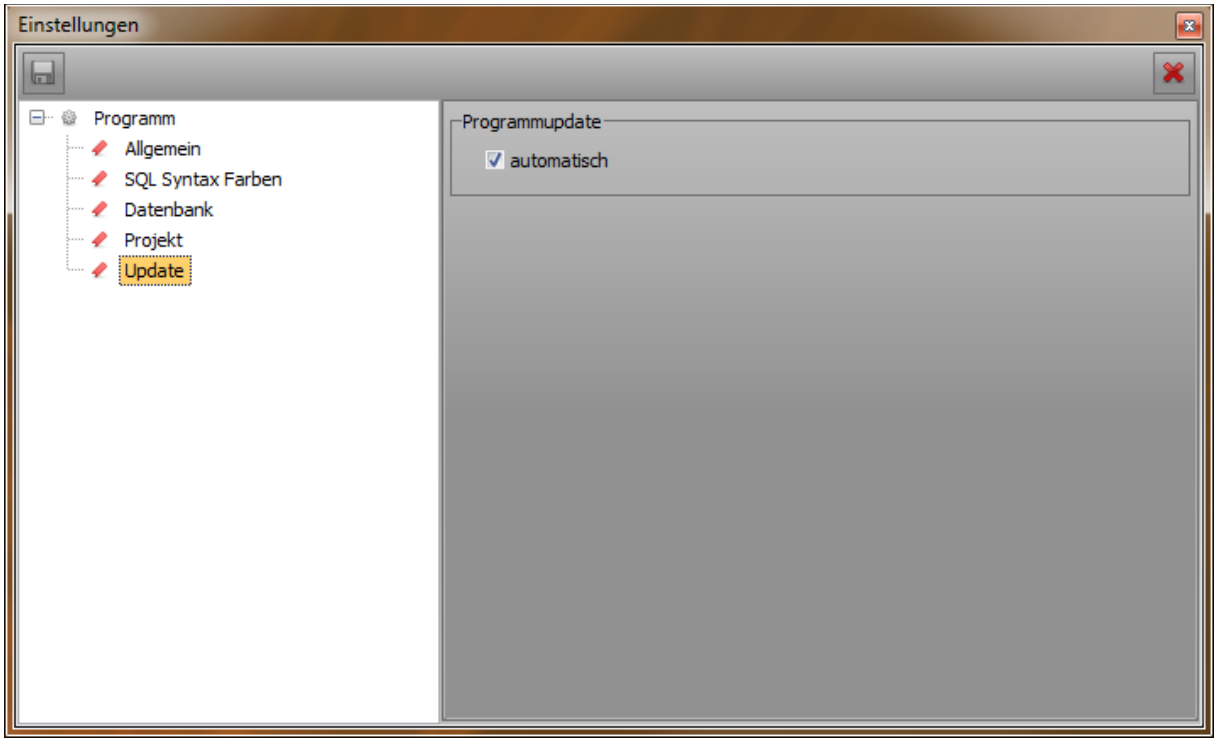
Hinweis:

Durch die zufällige Verschlüsselung ist der Inhalt der verschlüsselten SQL Datei nach jedem Speichern nie gleich. Deshalb sollten die verschlüsselten Dateien **nicht** ins Versionskontrollsystem aufgenommen werden.

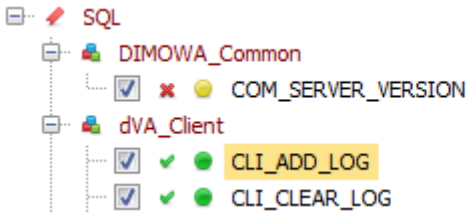
Ordner: Alle Ordner die zum Projekt gehören.

Datenbanken: Alle Datenbanken die zum Projekt gehören. Auswahl erfolgt aus der Datenbankliste.

2.8 Update

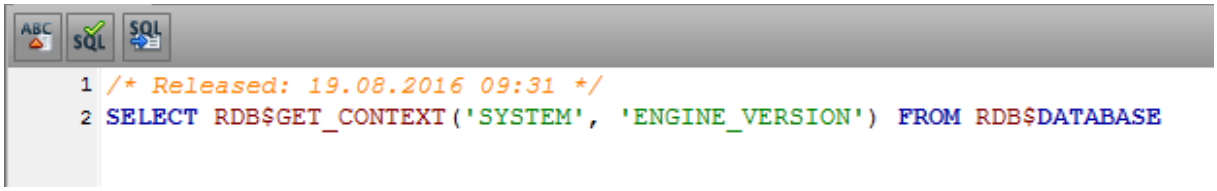


2.9 Status der SQL



- ✖ = **einer** der Statements ist „Closed" / „Gesperrt"
- ✔ = **alle** Statements sind „Released" / „Freigegeben"
- ● = **alle** Statements sind leer
- ● = **einer** der Statements ist leer
- ● = **alle** Statements sind ausgefüllt

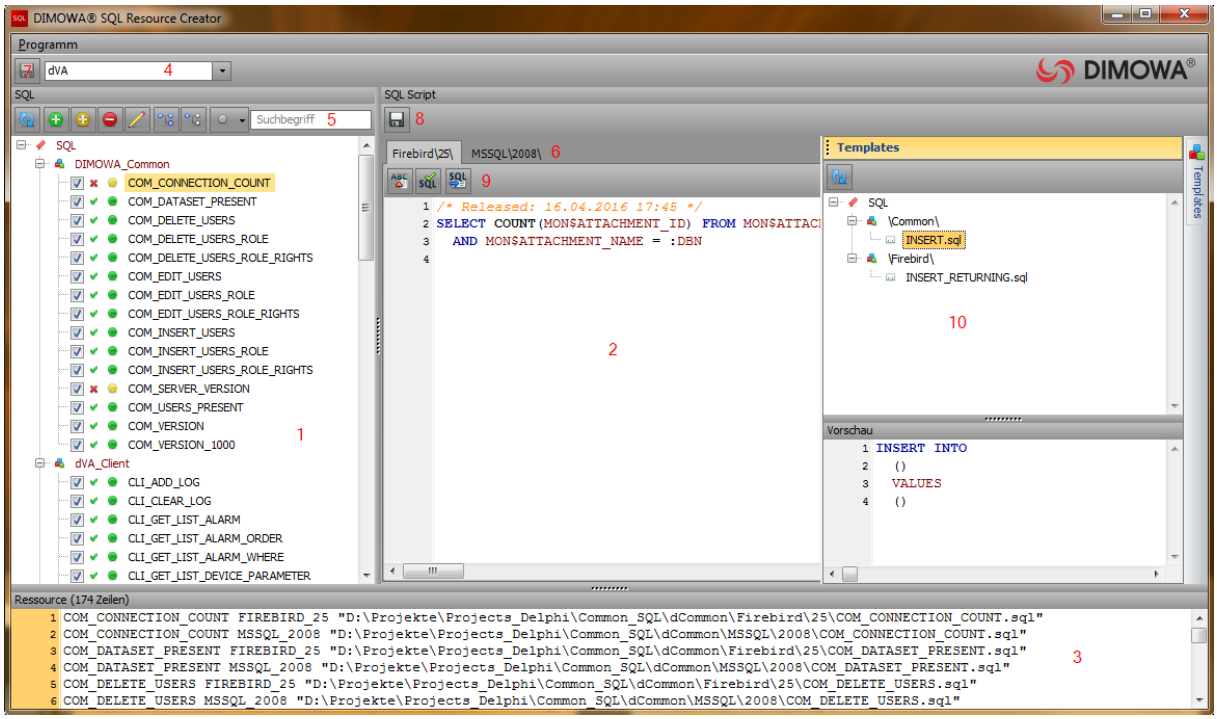
2.10 Header im SQL



Das SQL trägt den Status selbst damit Kollegen den aktuellen Status angezeigt bekommen. Beispielsweise bei Änderung des Release Status. Die Statuszeile wird automatisch neu erzeugt.

- = Wechsel des Status
- = komplettes Dokument nach UpperCase konvertieren
- = externen Editor aufrufen

2.11 GUI



1 = SQL List

2 = SQL Content

3 = Ressource Content (Read Only)

4 = Projekt Auswahl

5 = Suchbegriff

6 = verfügbare DMBS

7 = Projekt speichern

8 = SQL speichern

9 = Editorfunktionen

10 = Templates

2.12 Shortcuts

Shift+S	Projekt speichern
Ctrl+S	SQL speichern
Ctrl+Ins	SQL hinzufügen
Shift+Ins	SQL kopieren
Ctrl+Enf	SQL löschen
Ctrl+R	SQL umbenennen
Shift+Ctrl+C	Name des SQL in Zwischenablage kopieren
Ctrl+C	Kopieren
Ctrl+V	Einfügen
Ctrl+X	Ausschneiden
F5	Refresh SQL Baum
Ctrl+E	externen Editor aufrufen

3 Codebeispiele Delphi

3.1 Aufnehmen der Ressource

```
{$R *.res}
{$R 'dVA_SQL.res' '..\Ressources\dVA_SQL.rc'}

begin
  //ReportMemoryLeaksOnShutdown:= True;    --- macht jetzt
  Application.Initialize;
  Application.MainFormOnTaskbar := True;
```

3.2 Konstante für Datenbanken (einfacher)

```
conDatabaseResourceGroupString: array[0..1] of string = ('Firebird_25', 'MSSQL_2008');
```

3.3 Konstante für Key

```
conKey = '0815'; // besser einen sicheren Key verwenden. :-)
```

3.4 SQL aus Ressource extrahieren (Common) _Klartext

```
function TData GetSQLByName(SQLName: string): string;
var
  SQLStream: TResourceStream;
  SQLStrings: TStringList;
begin
  Result := '';
  SQLStrings := TStringList.Create;
  try
    SQLStream := TResourceStream.Create(HInstance, SQLName, PWideChar(conDatabaseResourceGroupString[FDatabaseProperties.DBMS]));
    try
      try
        SQLStrings.LoadFromStream(SQLStream);
        SQLStrings.Delete(0); // Kommentarzeile
        Result := SQLStrings.Text;
      except
        Result := '';
      end;
    finally
      SQLStream.Free;
    end;
  finally
    SQLStrings.Free;
  end;
end;
```

3.5 SQL aus Ressource extrahieren (Common) _Verschlüsselung

```
function T 3.GetSQLByName(SQLName: string): string;
var
  SQLStream: TResourceStream;
  SQLStrings: TStringList;
begin
  Result := '';
  SQLStrings := TStringList.Create;
  try
    SQLStream := TResourceStream.Create(HInstance, SQLName, PWideChar(conDatabaseResourceGroupString[FDatabaseProperties.DBMS]));
    try
      try
        SQLStrings.LoadFromStream(SQLStream);
        Result := TTools.Decrypt(SQLStrings.Text, conKey);
      except
        Result := '';
      end;
    finally
      SQLStream.Free;
    end;
  finally
    SQLStrings.Free;
  end;
end;
```

- eindeutige ID (Integer) für **FDatabaseProperties.DBMS** oder Alternative als Integer

3.6 TTools_Verschlüsselung (DEC 5.2)

```
var
  CipherMode: TCipherMode = cmCBCx;
  HashClass: TDECHashClass = THash_SHA256;
  TextFormat: TDECFormatClass = TFormat_MIME64;
  KDFIndex: LongWord = 1;

type
  TTools = class
  public
    class function Decrypt(aHash: string; aKey: string = ''): string;
    class function Encrypt(aText: string; aKey: string = ''): string;
  end;
```

```
class function TTools.Decrypt(aHash: string; aKey: string = ''): string;
var
  Cipher: TCipher_Rijndael;
  Salt: Binary;
  Data: Binary;
  Check: Binary;
  Pass: Binary;
  Len: Integer;
begin
  if aKey = '' then
  begin
    aKey := conKey;
  end;

  Cipher := TCipher_Rijndael.Create;
  try
    Salt := ValidFormat(TextFormat).Decode(RawByteString(aHash));
    Len := Length(Salt) - 16 - Cipher.Context.BufferSize;
    Data := Copy(Salt, 17, Len);
    Check := Copy(Salt, Len + 17, Cipher.Context.BufferSize);
    SetLength(Salt, 16);
    Pass := ValidHash(HashClass).KDFx(aKey[1], Length(aKey) * 2, Salt[1], Length(Salt), Cipher.Context.KeySize, TFormat_Copy, KDFIndex);
    Cipher.Mode := CipherMode;
    Cipher.Init(Pass);
    SetLength(Result, Len div 2);
    Cipher.Decode(Data[1], Result[1], Len);
    if Check <> Cipher.CalcMAC then
    begin
      Result := '';
      raise Exception.Create(conInvalidHash);
    end;
  finally
    Cipher.Free;
    ProtectBinary(Salt);
    ProtectBinary(Data);
    ProtectBinary(Check);
    ProtectBinary(Pass);
  end;
end;
```

```
class function TTools.Encrypt(aText: string; aKey: string = ''): string;
var
  Cipher: TCipher_Rijndael;
  Salt: Binary;
  Data: Binary;
  Pass: Binary;
begin
  if aKey = '' then
  begin
    aKey := conKey;
  end;

  Cipher := TCipher_Rijndael.Create;
  try
    Salt := RandomBinary(16);
    Pass := ValidHash(HashClass).KDFx(aKey[1], Length(aKey) * 2, Salt[1], Length(Salt), Cipher.Context.KeySize, TFormat_Copy, KDFIndex);
    Cipher.Mode := CipherMode;
    Cipher.Init(Pass);
    SetLength(Data, Length(aText) * 2);
    Cipher.Encode(aText[1], Data[1], Length(Data));
    Result := string(ValidFormat(TextFormat).Encode(Salt + Data + Cipher.CalcMAC));
  finally
    Cipher.Free;
    ProtectBinary(Salt);
    ProtectBinary(Data);
    ProtectBinary(Pass);
  end;
end;
```

3.7 SQL aus Ressource extrahieren ohne Parameter

```
procedure TData.GetServerVersion;
var
  Query: TIBCQuery;
begin
  Query := CreateQuery;
  try
    Query.SQL.Text := GetSQLByName('COM_SERVER_VERSION');
    Query.Open;
    FDatabaseProperties.ServerVersion := Query.Fields[0].AsString;
  finally
    Query.Free;
  end;
end;
```

3.8 SQL aus Ressource extrahieren mit Parameter

```
function TDa CreateLastRecordData(ParameterID: Integer): TRecordData;
var
  Qry: TIBCQuery;
begin
  Result := TRecordData.Create;
  Qry := CreateQuery;
  try
    Qry.SQL.Text := GetSQLByName('SER_CREATE_LAST_RECORD_DATA');
    Qry.ParamByName('PI').AsInteger := ParameterID;
    Qry.Open;
    if not Qry.Eof then
    begin
      Result.ParameterID := Qry.FieldByName('F_PARAMETER_ID').AsInteger;
      Result.ParameterType := ptUnknown;
      Result.RecordTime := UnixToDateTime(Qry.FieldByName('F_TIMESTAMP_UNIX').AsInteger);
      Result.PowerState := TDevicePowerState(Qry.FieldByName('F_POWER_STATE').AsInteger);
      Result.Measure := Qry.FieldByName('F_VALUE').AsString;
    end;
  finally
    Qry.Free;
  end;
end;
```

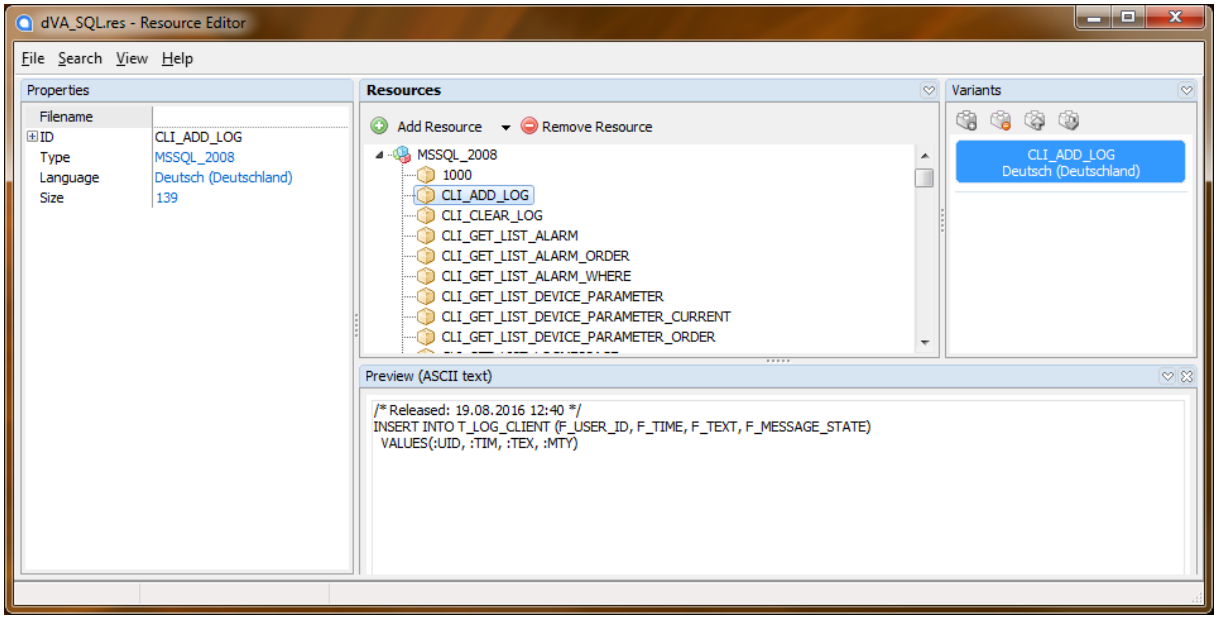
3.9 SQL aus Ressource extrahieren mit Parameter und Platzhaltern

```
procedure TData[redacted].SaveList(aList: TDeviceList);
var
  Qry: TIBCQuery;
  Proc: TIBCStoredProc;
  Device: TDevice;
  InList: string;

  procedure AddToInList(Value: Integer);
  begin
    InList := InList + IntToStr(Value) + ',';
  end;

begin
  Qry := CreateQuery;
  try
    Proc := CreateStoredProc;
    try
      Proc.StoredProcName := 'UI_DEVICES';
      Proc.Prepare;
      InList := '';
      for Device in aList do
      begin
        Proc.ParamByName('MASTER_DEVICE_ID').AsInteger := Device.MasterDeviceID;
        Proc.ParamByName('GROUP_ID').AsInteger := Device.Group.GroupID;
        Proc.ParamByName('ADRESS').AsInteger := Device.Adress;
        Proc.ParamByName('CAPTION').AsString := Device.Caption;
        Proc.ParamByName('NAME').AsString := Device.Name; // = mit ModelName
        Proc.ParamByName('ISACTIVE').AsInteger := 1;
        Proc.ExecProc;
        AddToInList(Proc.ParamByName('RET_ID').AsInteger);
      end;
      // Rücksetzen der nicht mehr aktiven
      Device := aList[0]; // ersten nehmen wegen ID
      InList := Copy(InList, 1, Length(InList) - 1); // letztes Komma entfernen
      Qry.SQL.Text := Format(GetSQLByName('SER_EDIT_DEVICES_ACTIVE'), [InList]);
      Qry.ParamByName('MDI').AsInteger := Device.MasterDeviceID;
      Qry.ExecSQL;
    finally
      Proc.Free;
    end;
  finally
    Qry.Free;
  end;
end;
```

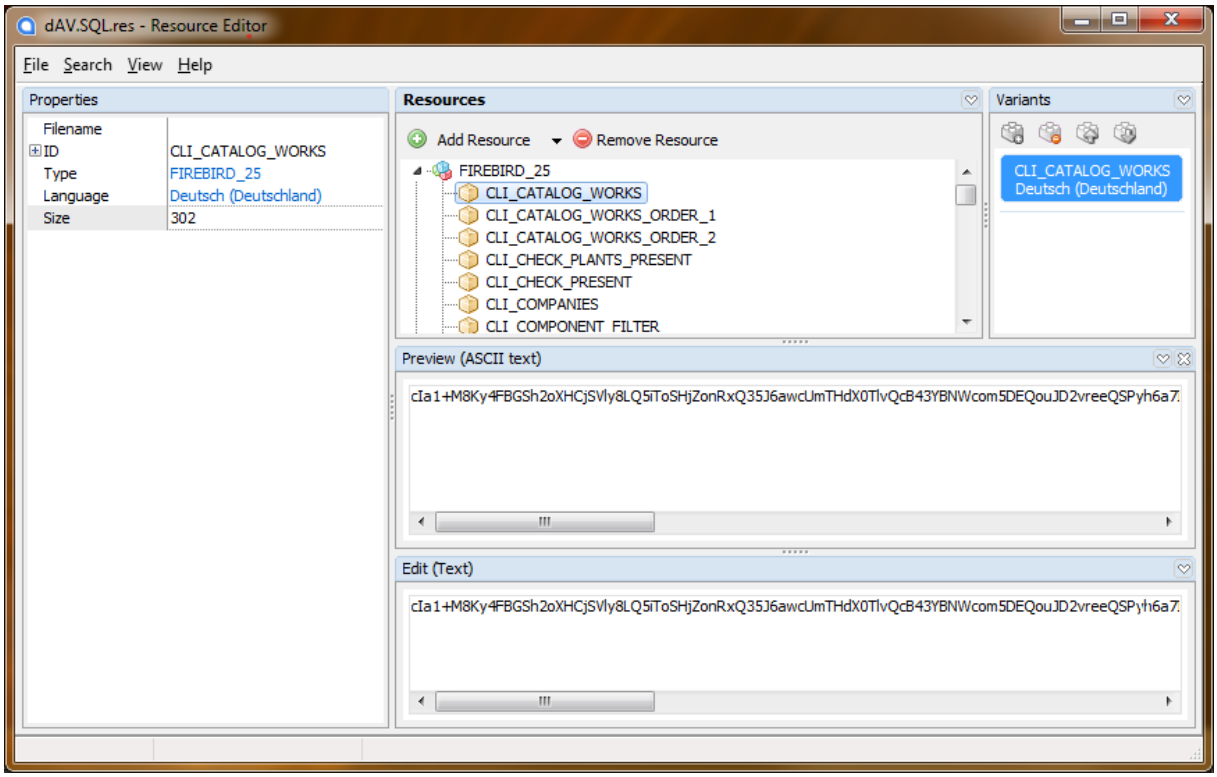
3.10 Compilieren

- WICHTIG: Projekt neu erzeugen damit die Änderungen an der *.rc Datei übernommen werden.

3.11 fertige Ressource (Klartext)



3.12 fertige Ressource (verschlüsselt)



4 Anhang / Ressourcen

- keine