

SQLITE3 MIT DELPHI XE3

... aus der Praxis für die Praxis ...

Robert W.B. Linn, Jahrgang 1958, befasst sich seit Mitte der 80er mit Delphi, damals noch auf DOS Basis mit Turbo Pascal, dann mit Turbo Delphi und aktuell mit Delphi XE3.

Impressum

Copyright: © 2013 Robert W.B. Linn, Pinneberg, Germany. Internet: <http://www.rwblinn.de>

Druck und Verlag: Kein

ISBN: Kein

1. Auflage 2013

Der Autor übernimmt keine Garantie und Haftung für eventuelle Fehler im Text und den Beispielen. Diesbezügliche Hinweise werden gerne entgegengenommen. Sollten Informationen (Texte und Grafiken) nicht auf dem neuesten Stand sein, wird hierfür keine Haftung übernommen.

SQLite ist [Public Domain](#). Die Marke SQLite ist beim [United States Patent and Trademark Office](#) registriert.

Delphi ist ein eingetragenes Warenzeichen von Embarcadero Technologies

Microsoft und Windows sind eingetragene Warenzeichen von Microsoft Corporation in den USA und anderen Ländern.

Die Bezeichnungen „Java“ und „JavaScript“ sind Warenzeichen oder eingetragene Warenzeichen von Sun Microsystems, Inc.

Andere verwendete Markennamen sind eingetragene Warenzeichen der jeweiligen Firmen und/oder Privatpersonen.

Inhaltsverzeichnis

Inhaltsverzeichnis	2
Einleitung.....	4
Information.....	4
SQLite Datentypen („Storage Classes“).....	4
Schema Beispieldatenbank	5
Schnelleinstieg.....	6
SQLite Datenbank und Tabelle erstellen	6
SQLite Datenbank Tabellenliste im TMemo	6
SQLite Datenbank Datensätze im TMemo	8
SQLite Datenbank Datensätze im TStringGrid.....	9
SQLite Datenbank Datensätze im DBGrid (Direkt)	10
SQLite Datenbank Datensätze im DBGrid (Aktionen)	11
SQLConnection	12
SQLConnection Vorbereiten.....	12
SQLConnection Eigenschaften.....	12
SQLConnection Verbindung öffnen.....	13
SQL-Befehle	14
SQL-Befehle ausführen - SQLConnection	14
SQL-Befehle ausführen - SQLDataSet.....	15
SQL-Befehl Beispiele.....	16
CREATE TABLE Tabelle (Spalte1 Typ, SpalteN Typ)	16
INSERT INTO Tabelle (Spalte1, SpalteN) VALUES (Wert1, WertN).....	17
INSERT INTO Tabelle (Spalte1, SpalteN) VALUES (:Spalte1, :SpalteN)	18
SELECT Spalten FROM Tabelle(n) WHERE Bedingung ORDER BY Sortierung GROUP BY Gruppierung HAVING Einschränkung	19
UPDATE Tabelle SET Spalte1=Wert, SpalteN=Wert WHERE Spalte=Bedingung	20
DELETE FROM TABLE WHERE Spalte=Bedingung.....	20
DROP TABLE Tabellenname.....	20
CREATE INDEX Indexname ON Tabelle(Spalte)	20
DROP INDEX Indexname.....	20
VIEWS	21
CREATE VIEW Name AS SELECT Befehl.....	21
DROP VIEW Viewname.....	21
TRIGGER.....	22
CREATE TRIGGER Name AFTER Aktion ON Tabelle BEGIN Ereignis; END.....	22
Tabelle Log erstellen.....	22
Trigger erstellen	23

PRAGMA	24
PRAGMA database_list.....	24
PRAGMA table_info (Tabellenname)	24
PRAGMA index_list (Tabellenname)	25
PRAGMA index_info (Indexname).....	25
PRAGMA auto_vacuum	25
PRAGMA integrity_check	25
SQLite SELECT Beispiele.....	26
SELECT ausführen	26
SELECT Ergebnis im StringGrid darstellen	27
SELECT Ergebnis TLabel und TEdit zuweisen	28
SELECT Ergebnis TStringList und TListBox zuweisen	28
SELECT Ergebnis in HTML Tabelle ausgeben	29
SELECT auf mehrere Tabellen (JOIN).....	30
SELECT sqlite_master	31
SELECT CAST (Ausdruck AS Datentyp [(Länge)])	31
SQLite und FireDAC Einstieg.....	32
SQLite spezielle Anwendungsbeispiele	33
SQLite Datensätze im DBGRID.....	33
SQLite BLOBs verwenden	36
SQLite SQL-Befehle direkt ausführen	38
SQLite Tabelle für Einstellungen verwenden	39
Anhang	41
Fehlermeldungen und Abhilfe (unvollständige Liste!)	41
Zusammenfassung einige SQL-Befehle (Beispiele).....	42
Referenz Beispieldatenbank Buecher.db SQL-Befehle.....	43

Einleitung

Die vorliegende Anleitung vermittelt den praxisnahen Umgang mit SQLite3 unter [Embarcadero Delphi XE3](#). Ziel ist es anhand Beispiele die direkte Anwendung zu zeigen.

SQLite ist ein relationales Datenbanksystem, welches [SQL-92-Standard](#) festgelegten SQL-Sprachbefehle weitgehend unterstützt.

SQLite wird in allen Bereiche eingesetzt und ist als Datenbank nicht mehr wegzudenken.

SQLite ist [Public Domain](#).

Information

- Information SQLite im Internet unter [SQLite Org](#)
- Zur Anleitung verschiedene [Projektbeispiele](#) mit [Quellcode](#) Delphi XE3:
 - SQLite DBGrid verwenden (ropDelphiSQLiteDBGrid)
 - SQLite Blobs verwenden (ropDelphiSQLiteBlob)
 - SQLite SQL-Befehle direkt ausführen (ropSQLiteIt)
 - Mini Buecherverwaltung (ropBookLib)

SQLite Datentypen („Storage Classes“)

Datentyp	Wert	(Priorität) Typ-Affinität
NULL	NULL	
INTEGER	Eine Ganzzahl, gespeichert in 1, 2, 3, 4, 6, oder 8 Byte je nach Größe des Wertes (signed 64-bit)	(1) INT, INTEGER, TINYINT, SMALLINT, MEDIUMINT, BIGINT, UNSIGNED BIG INT, INT2, INT8, FLOATING POINT
TEXT	Ein String, gespeichert unter Verwendung der Datenbank-Kodierung (UTF-8, UTF-16BE oder UTF-16LE) (Max Grösse Vorgabe 1,000,000,000 bytes)	(2) CHARACTER(20), VARCHAR(255), VARYING CHARACTER(255), NCHAR(55), NATIVE CHARACTER(70), NVARCHAR(100), TEXT, CLOB
BLOB	Eine Masse von Daten, gespeichert wie eingegeben	(3) BLOB
REAL	Eine Fließkommazahl gespeichert als IEEE Gleitkommazahl von 8 Byte (64-bit (double))	(4) REAL, DOUBLE, DOUBLE PRECISION, FLOAT
NUMERIC		(5) NUMERIC, DECIMAL(10,5), BOOLEAN, DATE, DATETIME, STRING

HINWEIS

- Die **Typ-Affinität** einer Spalte ist der empfohlene Datentyp für in dieser Spalte gespeicherte Daten.

Schema Beispieldatenbank

Für diese Anleitung wird die Datenbank **BUECHER.DB** bestehend aus den 3 Tabellen **Buecher**, **Autoren**, **Log** verwendet.

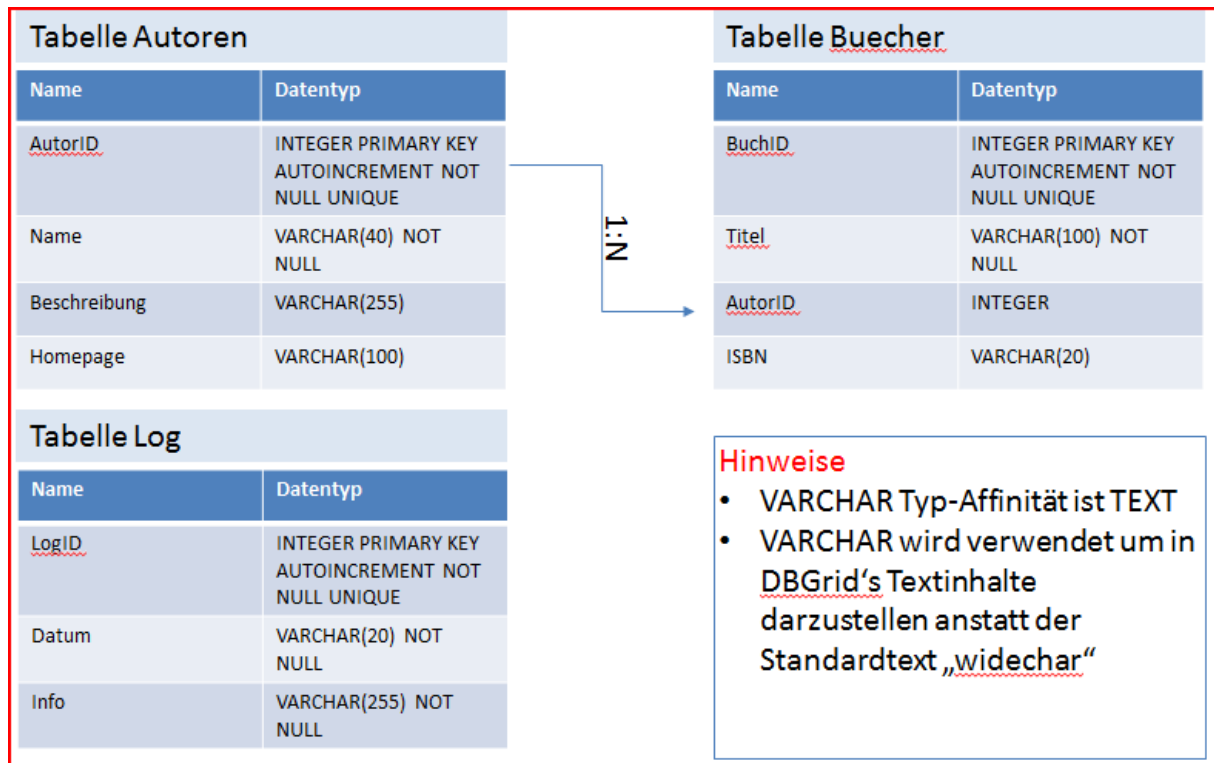


Abbildung 1

Schnelleinstieg

SQLite Datenbank und Tabelle erstellen

Eine neue SQLite Datenbank Buecher.db mit Tabelle Buecher erstellen.

Aktivität	Beschreibung
Delphi Projekt erstellen oder öffnen, Form wählen	Vgl. frmMain
TSQLConnection Komponente hinzufügen	TSQLConnection (dbExpress). Eigenschaften anpassen: Name: SQLConnection; Driver=SQLite; LoginPrompt=False
Variable definieren	<pre>Var FSQLErrorText: String; FSQLErrorDataSet: TDataSet; sStr: String; i: Integer;</pre>
SQLite Datenbank erstellen	<pre>Try SQLConnection.Connected := False; SQLConnection.Params.Values['Database'] := '<Pfad>\Buecher.db'; SQLConnection.Params.Values['FailIfMissing'] := 'False'; SQLConnection.Connected := True; Except On E: EDatabaseError Do ShowMessage(E.Message); End;</pre>
Tabelle hinzufügen	<pre>Try FSQLErrorText := 'CREATE TABLE Buecher (BuchID INTEGER PRIMARY KEY AUTOINCREMENT NOT NULL UNIQUE, Titel VARCHAR(100) NOT NULL, AutorID INTEGER, ISBN VARCHAR(20));'; SQLConnection.ExecuteDirect(FSQLErrorText); Except On E: EDatabaseError Do ShowMessage(E.Message); End;</pre>
Testdaten hinzufügen	<pre>Try FSQLErrorText := 'INSERT INTO Buecher (BuchID, Titel, AutorID, ISBN) VALUES (NULL, "B1", 1, "123");'; SQLConnection.ExecuteDirect(FSQLErrorText); FSQLErrorText := 'INSERT INTO Buecher (BuchID, Titel, AutorID, ISBN) VALUES (NULL, "B2", 1, "456");'; SQLConnection.ExecuteDirect(FSQLErrorText); Except On E: EDatabaseError Do ShowMessage(E.Message); End;</pre>
Testdaten zeigen	<pre>Try FSQLErrorText := 'SELECT * FROM Buecher;'; SQLConnection.Execute(FSQLErrorText, NIL, FSQLErrorDataSet); FSQLErrorDataSet.First; While NOT FSQLErrorDataSet.EOF Do Begin sStr := ''; For i := 0 To FSQLErrorDataSet.FieldCount - 1 Do sStr := sStr + ' ' + FSQLErrorDataSet.Fields[i].AsString; ShowMessage(sStr); FSQLErrorDataSet.Next; End; Except On E: EDatabaseError Do ShowMessage(E.Message); End;</pre>


SQLite Datenbank Tabellenliste im TMemo

Aktivität	Beschreibung
Delphi Projekt erstellen oder öffnen, Form wählen	Vgl. frmMain
Komponenten hinzufügen	TSQLConnection (dbExpress). Eigenschaften anpassen: Name: SQLConnection; Driver=SQLite; LoginPrompt=False TMemo (Standard). Eigenschaften anpassen: Name: memoOutput; ScrollBars: ssBoth; WordWrap: False;
Procedure FormCreate erstellen/anpassen	SQLConnection Pfad zur bestehenden SQLite Datenbank Buecher.db setzen und verbinden. Ist die Datenbank nicht vorhanden, dann Abbruch. <div data-bbox="571 595 1370 819" style="border: 1px solid black; padding: 5px;"> <pre> Try SQLConnection.Connected := False; SQLConnection.Params.Values['Database'] := '<SQLiteDBPfad>'; SQLConnection.Params.Values['FailIfMissing'] := 'True'; SQLConnection.Connected := True; Except On E: EDatabaseError Do ShowMessage(E.Message); End; </pre> </div>
Procedure FormCreate erweitern	Datenbank ist geöffnet, Tabellenliste im TMemo ausgeben: <div data-bbox="571 909 1370 1480" style="border: 1px solid black; padding: 5px;"> <pre> Var FSQLErrorText: String; FSQLErrorDataSet: TDataSet; sLine: String; i: Integer; Try memoOutput.Lines.Text := 'Tabellen: '; FSQLErrorText := 'SELECT name FROM SQLITE_MASTER WHERE TYPE="table" ORDER BY name;'; SQLConnection.Execute(FSQLErrorText, NIL, FSQLErrorDataSet); FSQLErrorDataSet.First; While NOT FSQLErrorDataSet.EOF Do Begin sLine := ''; For i := 0 To FSQLErrorDataSet.FieldCount - 1 Do sLine := sLine + FSQLErrorDataSet.Fields[i].AsString; memoOutput.Lines.Add(sLine); FSQLErrorDataSet.Next; End; Except On E: EDatabaseError Do ShowMessage(E.Message); End; </pre> </div>
Inhalt TMemo	Ergebnis SQL-Abfrage: SELECT name FROM SQLITE_MASTER WHERE TYPE="table" ORDER BY name; <div data-bbox="544 1565 868 1686" style="border: 1px solid red; padding: 5px;"> <pre> Tabellen: Autoren Buecher Log sqlite_sequence </pre> </div> <p>Abbildung 2</p>

SQLite Datenbank Datensätze im TMemor

Aktivität	Beschreibung
SQLite Datenbank öffnen	Siehe vorher.
SQL-Abfrage ausführen und Ergebnis speichern (TDataSet)	<pre> Var FSQLErrorText: String; FSQLErrorCode: TDataSet; FSQLErrorText := 'SELECT * FROM Autoren;'; SQLConnection.Execute(FSQLErrorText, NIL, FSQLErrorCode); </pre>
Datensätze im TMemo zeigen	<pre> Var FSQLErrorText: String; FSQLErrorCode: TDataSet; sLine: String; i: Integer; memoSQLiteDatabaseQueryResult.Lines.Text := 'Anzahl Datensätze: ' + IntToStr(FSQLErrorCode.RecordCount); // Spaltennamen sLine := ''; For i := 0 To FSQLErrorCode.FieldCount - 1 Do sLine := sLine + ' ' + FSQLErrorCode.Fields[i].FieldName; memoSQLiteDatabaseQueryResult.Lines.Add(sLine); memoSQLiteDatabaseQueryResult.Lines.Add('-----'); // Inhalt FSQLErrorCode.First; While NOT FSQLErrorCode.EOF Do Begin sLine := ''; For i := 0 To FSQLErrorCode.FieldCount - 1 Do sLine := sLine + ' ' + FSQLErrorCode.Fields[i].AsString; memoSQLiteDatabaseQueryResult.Lines.Add(sLine); FSQLErrorCode.Next; End; </pre>
Inhalt TMemo	<p>Ergebnis SQL-Abfrage: SELECT * FROM Autoren</p> <div style="border: 2px solid red; padding: 5px;"> <p>Anzahl Datensätze: 3</p> <p>AutorID Name Beschreibung Homepage</p> <p>-----</p> <p>1 Hannes Nygaard</p> <p>2 Derek Meister</p> <p>3 Adler Olsen</p> </div> <p>Abbildung 3</p>

SQLite Datenbank Datensätze im TStringGrid

Aktivität	Beschreibung
SQLite Datenbank öffnen	Siehe vorher.
Form TStringGrid Komponente hinzufügen	TStringGrid (Additional). Eigenschaften anpassen: Name sgSQLiteDBQueryToStringGrid Options: [goFixedVertLine, goFixedHorzLine, goVertLine, goHorzLine, goRangeSelect, goColSizing, goRowSelect]
SQL-Abfrage ausführen	<pre> Var FSQSqlCommandText: String; FSQDataSet: TDataSet; FSQSqlCommandText := 'SELECT * FROM Autoren;'; SQLConnection.Execute(FSQSqlCommandText, NIL, FSQDataSet); </pre>
Datensätze lesen und im StringGrid anzeigen	<pre> If FSQDataSet.IsEmpty Then Exit; StringGrid1.ColCount := FSQDataSet.FieldCount + 1; StringGrid1.RowCount := FSQDataSet.RecordCount + 1; StringGrid1.FixedCols := 1; StringGrid1.FixedRows := 1; StringGrid1.ColWidths[0] := 16; //Fixspalte Breite anpassen // Spaltenüberschriften setzen For nCol := 0 To FSQDataSet.FieldCount - 1 Do StringGrid1.Cells[nCol + 1, 0] := FSQDataSet.Fields[nCol].FieldName; nLin := 0; // Daten im Stringgrid zeigen FSQDataSet.First; While Not FSQDataSet.Eof Do Begin For nCol := 0 To FSQDataSet.FieldCount - 1 Do StringGrid1.Cells[nCol+1,nLin + StringGrid1.FixedRows] := FSQDataSet.Fields[nCol].AsString; FSQDataSet.Next; Inc(nLin); End; </pre>
Inhalt TStringGrid	 <p>Abbildung 4</p>

SQLite Datenbank Datensätze im DBGrid (Direkt)

Eine vorhandene SQLite Datenbank öffnen und die Datensätze in einem DBGrid zeigen.

Aktivität	Beschreibung
Verzeichnis für ein neues Delphi Projekt erstellen	Vgl. Buecher
Dateien in das neue Verzeichnis kopieren	DLL sqlite3.dll (oder in das Windows System Verzeichnis kopieren) Eine bestehende Datenbank kopieren. Vgl. buecher.db
Delphi neues Projekt erstellen	File > New > VCL Forms Application Delphi
Projekt Eigenschaften anpassen	Project > Options > Delphi Compiler Target: Release Configuration; Output Directory: Leeren Eintrag; Unit Output Directory: Leeren Eintrag
Form Form1 Eigenschaften anpassen	Name: frmMain; Caption: Bücherliste; Position: poDesktopCenter
In Form frmMain Komponenten hinzufügen verbinden und anpassen	TSQLConnection (dbExpress) -> TSQLDataSet (dbExpress) -> TDataSetProvider (Data Access)-> ClientDataSet (Data Access)-> TDataSource (Data Access) -> TDBGrid (Data Controls)
TSQLConnection	Name: SQLConnection; Driver=SQLite; LoginPrompt=False; Params: FailIfMissing=False, Database=buecher.db; Connected=True
TSQLDataset Und SQL-Befehl definieren	Name: SQLDataSet; Connection: SQLConnection; CommandText=SELECT * FROM Buecher; CommandType=ctQuery; Active=True
TDataSetProvider	Name=DataSetProvider; Dataset=SQLDataSet
TClientDataSet	Name=ClientDataSet ; Providername=DataSetProvider; Active=True
TDataSource	Name=DataSource; DataSet=ClientDataSet
TDBGrid	Name=DBGrid; DataSource=DataSource; ReadOnly=True; Options=RowSelect zusätzlich aktivieren; Align=alClient; Columns=Add All Fields, ggf. Feldeigenschaften anpassen, wie Titel (Title), Spaltenbreite (Width)
Projekt kompilieren und ausführen (F9)	Unit speichern unter umain Projekt speichern unter ropdelphisqliteeinstieg

Design

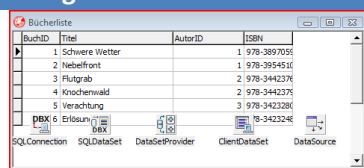


Abbildung 5

Run

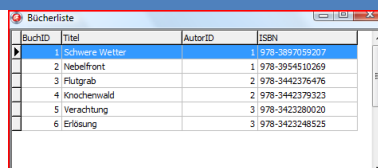


Abbildung 6

Run (Spaltentitel angepasst)

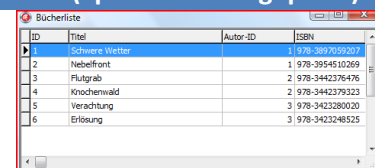


Abbildung 7

HINWEIS

- Formular entsprechend erweitern, wie z.B. TDBNavigator Komponente, TEdit für SQL-Befehl SELECT.

SQLite Datenbank Datensätze im DBGrid (Aktionen)

Eine vorhandene SQLite Datenbank öffnen und die Datensätze in einem DBGrid zeigen.

Verwendet werden Aktionen (Actions).

Form und Komponente haben den gleichen Aufbau, wie im vorherigen Beispiel.

Aktivität	Beschreibung Aktion (Action)
SQLite Datenbank verbinden	<pre> Procedure TfrmMain.ActionDBConnectExecute(Sender: TObject); Begin Try SQLConnection.Connected := False; SQLDataSet.Active := Connected; SQLConnection.Params.Values['Database'] := '<SQLite DB Pfad>'; SQLConnection.Params.Values['FailIfMissing'] := 'False'; SQLConnection.Connected := True; Except On E: EDatabaseError Do ShowMessage(E.Message); End; End;</pre>
SQL-Abfrage ausführen und Datensätze im TDBGrid zeigen	<pre> Procedure TfrmMain.ActionQueryToDBGridExecute(Sender: TObject); Var FSQLCommandText : String; Begin // Query aus einem TMemo verwenden FSQLCommandText := memoSQLiteDBQuery.Text; // Query prüfen ob SELECT Befehl If (LeftStr(LowerCase(FSQLCommandText),6) <> 'select') Then Begin ShowMessage('Abbruch: SQL-Query enthält kein SELECT Befehl!'); Exit; End; // SQLite DB verbinden If SQLConnection.Connected = False Then ActionSQLiteDBConnectExecute(Sender); If SQLConnection.Connected = False Then Exit; Try // ClientDataSet and SQLDataSet deaktivieren ClientDataSet.Active := False; SQLDataSet.Active := False; // SQL-Befehl zuweisen SQLDataSet.CommandText := FSQL; // ClientDataSet and SQLDataSet aktivieren SQLDataSet.Active := True; ClientDataSet.Active := True; // Query ausführen SQLDataSet.Open; Except On E: EDatabaseError Do ShowMessage(E.Message); End; End;</pre>

SQLConnection

Die Verbindung mit einer SQLite Datenbank (Datei) erfolgt über die Komponente **dbExpress TSQLConnection**.

SQLConnection Vorbereiten

- Delphi Projekt neu erstellen oder öffnen
- VCL Formular neu erstellen oder wählen
- Komponente hinzufügen: **dbExpress TSQLConnection**



Abbildung 8

- Dataset hinzufügen um das Ergebnis eines SQL SELECT Befehls zu speichern.

BEISPIELE

Var FDSSQLResults: TDataSet oder eine TSQLDataSet Komponente.

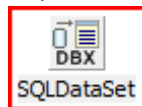


Abbildung 9

WICHTIG

- Die Datei **sqlite3.dll** im Projekt Verzeichnis ablegen oder im Windows System Verzeichnis.

SQLConnection Eigenschaften

- Name: SQLConnection
- Connected: False (noch keine Verbindung erstellen)
- Driver: SQLite (nicht vergessen anzugeben)
- Connectionname: SQLConnect (kann beliebig sein)
- KeepConnection: True (Verbunden bleiben nach die Verbindung erstellt wurde)
- Params: FailIfMissing=False (kein Abbruch falls die Datenbank nicht gefunden wurde. Die Datenbank wird dann beim SQL-Befehl Create Table neu erstellt.

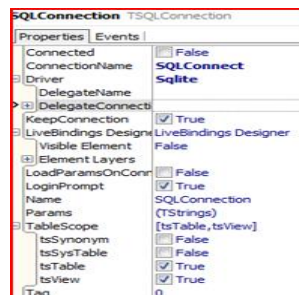


Abbildung 10

HINWEIS

- Parameter im Code definieren mittels Parameter=Wert (OHNE Leerzeichen)

```
SQLConnection.Params.Clear;
SQLConnection.Params.Add('Database=' + 'buecher.db');
SQLConnection.Params.Add('FailIfMissing=False');
```

oder

```
SQLConnection.Params.Values['Database'] := 'buecher.db';
SQLConnection.Params.Values['FailIfMissing'] := 'False';
```

SQLConnection Verbindung öffnen

Die Verbindung öffnen, nachdem der Datenbankpfad als Parameter definiert wurde. Wird nur der Dateiname angegeben, dann muss die Datei im Projektverzeichnis vorhanden sein.

```
Try
    // Datenbank angeben - kann noch erweitert werden mit z.B. Pfadangabe
    SQLConnection.Params.Add('Database=' + 'buecher.db');
    // Verbinden
    SQLConnection.Connected := True;
Except On E: EDatabaseError Do ShowMessage(E.Message);
End;
```

HINWEISE

- Datenbankdatei prüfen ob vorhanden:

```
If NOT FileExists('buecher.db') Then Abbruch;
```

- Datenbankname Pfad hinzufügen:

```
ExtractFilePath(Application.ExeName) + 'buecher.db';
oder
ExtractFilePath(ParamStr(i)) + 'buecher.db';
```

SQL-Befehle

SQL-Befehle ausführen - SQLConnection

Es gibt zwei Arten von SQL-Befehlen.

Mit Ergebnis: SQL-Befehl `SELECT` liefert Datensätze in einem `TDataset`.

```
SQLConnection.Execute(SQL-Befehl, NIL, TDataset);
```

Ohne Ergebnis: SQL-Befehle wie `CREATE`, `INSERT`, `DELETE`, `UPDATE`, `DROP` usw..

```
SQLConnection.Execute(SQL-Befehl, NIL);  
oder  
SQLConnection.ExecuteDirect(SQL-Befehl);
```

BEISPIEL SELECT

Alle Datensätze der Tabelle **Buecher** in einem Dataset speichern. Fehler mit `Try-Except` abfangen.

```
Var FDSSQLResults: TDataset;      // Ergebnis SQL-Befehl SELECT speichern  
Try  
    // Datensätze der Tabelle Buecher selektieren und in einem Dataset speichern  
    SQLConnection.Execute('SELECT * FROM Buecher', NIL, FDSSQLResults);  
Except On E: EDatabaseError Do ShowMessage(E.Message);  
End;
```

SQL-Befehle ausführen - SQLDataSet

Wird ein SQLDataSet verwendet, dann werden SQL-Befehle wie folgt ausgeführt:

Mit Ergebnis: SQL-Befehl `SELECT` liefert Datensätze in dem SQLDataSet.

```
SQLDataSet.CommandText := 'SQL SELECT Befehl';  
SQLDataSet.Open;
```

Ohne Ergebnis: SQL-Befehle wie `CREATE`, `INSERT`, `DELETE`, `UPDATE`, `DROP` usw..

```
SQLDataSet.CommandText := 'SQL CREATE oder INSERT ... Befehl';  
SQLDataSet.ExecSQL;
```

BEISPIEL SELECT

Alle Datensätze aus der Tabelle **Buecher** selektieren und in einem Stringgrid **sgSQLiteDBQueryToStringGrid** darstellen.

```
Procedure TfrmMain.SQLiteQueryToStringGrid(Sender: TObject);  
  Var i: Integer;  
Begin  
  Try  
    FSQLCommandText := 'SELECT * FROM Buecher;';  
    // Alle Informationen aus der Tabelle Buecher lesen  
    FSQLDataSet.Open;  
    // Spaltenzahl anpassen  
    sgSQLiteDBQueryToStringGrid.ColCount := FSQLDataSet.FieldCount + 1;  
    // Zeilenzahl anpassen  
    sgSQLiteDBQueryToStringGrid.RowCount := FSQLDataSet.RecordCount + 1;  
    For i := 0 To FSQLDataSet.FieldCount - 1 Do  
      // Spaltenüberschriften  
      SgSQLiteDBQueryToStringGrid.Cells[i + 1,0] := FSQLDataSet.Fields[i].FieldName;  
    i := 1;  
    FSQLDataSet.First;  
    // Daten aus Dataset im StringGrid übertragen  
    While NOT FSQLDataSet.EOF Do Begin  
      sgSQLiteDBQueryToStringGrid.Cells[1,i] := FSQLDataSet.Fields[0].AsString;  
      // Zelle Spalte 1, Zeile i  
      sgSQLiteDBQueryToStringGrid.Cells[2,i] := FSQLDataSet.Fields[1].AsString;  
      // Zelle Spalte 2, Zeile i  
      FSQLDataSet.Next;  
      i := i + 1;  
    End;  
  Except on E: EDatabaseError do  
    ShowMessage(E.Message);  
  End;  
End;
```

SQL-Befehl Beispiele

Einige Beispiele anhand der Tabelle **Buecher** aus der Datenbank **BUECHER.DB**.

HINWEISE

- In den Beispielen werden SQL-Befehle im String LSQl gespeichert.
- Groß-/Kleinschreibung bei SQL-Befehlen beachten.

CREATE TABLE Tabelle (Spalte1 Typ, SpalteN Typ)

Beispiel: Eine neue Tabelle **Buecher** erstellen.

HINWEIS

- SQLite Datentypen („Storage Classes“) : NULL, INTEGER, REAL, TEXT, BLOB

SQL-Befehle um die Tabellen Autoren und Buecher zu erstellen:

Var LSQlCreateAutoren, LSQlCreateBuecher, LSQlCreateLog: String;

```
LSQlCreateAutoren := 'CREATE TABLE Autoren (AutorID INTEGER PRIMARY KEY AUTOINCREMENT NOT NULL UNIQUE, Name VARCHAR(40) NOT NULL, Beschreibung VARCHAR(255), Homepage VARCHAR(100));';
```

```
LSQlCreateBuecher := 'CREATE TABLE Buecher (BuchID INTEGER PRIMARY KEY AUTOINCREMENT NOT NULL UNIQUE, Titel VARCHAR(100) NOT NULL, AutorID INTEGER, ISBN VARCHAR(20));';
```

```
LSQlCreateLog := 'CREATE TABLE Log (LogID INTEGER PRIMARY KEY AUTOINCREMENT NOT NULL UNIQUE, Datum VARCHAR(20) NOT NULL, Info VARCHAR(255) NOT NULL);';
```

SQLConnection Execute Möglichkeit:

```
SQLConnection.Execute(LSQlCreateAutoren,NIL);  
SQLConnection.Execute(LSQlCreateBuecher,NIL);  
SQLConnection.Execute(LSQlCreateLog,NIL);
```

HINWEISE

- Tabelle erstellen, falls nicht vorhanden: CREATE TABLE IF NOT EXISTS ...
- Anstatt SQLConnection.Execute(SQL-Befehl) kann auch SQLConnection.ExecuteDirect(SQL-Befehl) eingesetzt werden
- Namen für Tabellen und Spalten können auch in Anführungszeichen angegeben werden. Vgl. "Buecher".
- Datentyp TEXT wird in DBGRID als Memo dargestellt.
Um eine Zeichenkette (String) darzustellen, VARCHAR(Länge) verwenden.
Vgl. VARCHAR(20) entspricht einer Zeichenkette mit der Länge von 20 Zeichen.

SQLDataSet ExecSQL Möglichkeit:

```
SQLDataSet.CommandText := LSQlCreateAutoren;  
SQLDataSet.ExecSQL;  
SQLDataSet.CommandText := LSQlCreateBuecher;  
SQLDataSet.ExecSQL;  
SQLDataSet.CommandText := LSQlCreateLog;  
SQLDataSet.ExecSQL;
```


INSERT INTO Tabelle (Spalte1, SpalteN) VALUES (Wert1, WertN)

BEISPIEL: DATENSÄTZE IN DIE TABELLE BUECHER EINFÜGEN.

Via `SQLConnection.ExecuteDirect`

```
SQLConnection.ExecuteDirect('INSERT INTO Buecher(Titel) VALUES("Titel 1");');  
SQLConnection.ExecuteDirect('INSERT INTO Buecher(Titel, AutorID) VALUES("Titel 2",  
1);');
```

Via einem `TSQLDataSet` (siehe auch weiter unten) Testdaten mittels For Schleife erstellen:

```
For i := 1 To 5 Do Begin  
  SQLDataset1.CommandText :=  
    'INSERT OR REPLACE INTO Buecher (BuchID, Titel, AutorID, ISBN) ' +  
    'VALUES(NULL, "Titel ' + IntToStr(i) + '", "' + IntToStr(i) + '", "' + IntToStr(i) +  
    '");';  
  SQLDataset1.ExecSQL;  
End;
```

INSERT INTO Tabelle (Spalte1, SpalteN) VALUES (:Spalte1, :SpalteN)

Datensätze können in die Tabelle **Buecher** auch durch Angabe von Parametern eingefügt werden. Die Verwendung von Parametern, :Spalte1 .. :SpalteN, vereinfacht die Handhabung der Zuweisung von Werten.

BEISPIEL SQLCONNECTION > SQLDATASET > STRINGGRID

SQLDataSet definieren:

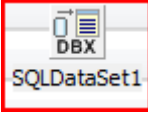


Abbildung 11

SQLDataSet1 ist mit
SQLConnection1 verbunden.

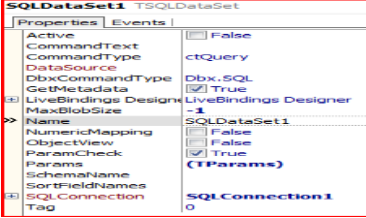


Abbildung 12

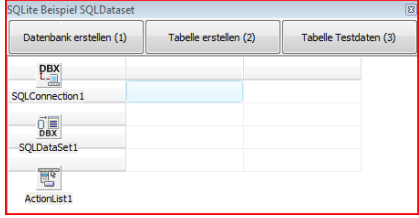


Abbildung 13

Tabelle Buecher Spalten mit Testdaten füllen:

```

For i := 1 To 5 Do Begin
    // Parameter definieren (Spaltennamen verwenden)
    CommandText := 'INSERT INTO Buecher (BuchID, Titel, AutorID, ISBN) VALUES(:BuchID,
:Titel, :AutorID, :ISBN)';
    // Parameter Werte zuweisen (Spaltennamen verwenden)
    SQLDataset1.ParamByName('BuchID').Value := NULL;
    SQLDataset1.ParamByName('Titel').Value := 'Mein Buch ' + IntToStr(i);
    SQLDataset1.ParamByName('AutorID').Value := i;
    SQLDataset1.ParamByName('ISBN').Value := 'ISBN' + IntToStr(i);
    // Und Werte in der Tabelle Buecher einfügen
    SQLDataset1.ExecSQL;
End;

```

Ausgabe im StringGrid:

```

CommandText := 'SELECT * FROM Buecher;';
SQLDataset1.Open;
StringGrid1.ColCount := SQLDataset1.FieldCount + 1; // Spalten
StringGrid1.RowCount := SQLDataset1.RecordCount + 1; // Zeilen
// Spaltenüberschrift
For i := 0 To FieldCount - 1 Do
    StringGrid1.Cells[i + 1,0] := SQLDataset1.Fields[i].FieldName;
// Werte im Stringgrid ausgeben
i := 1;
SQLDataset1.First;
While NOT SQLDataset1.EOF Do Begin
    // Spalte Nr
    SQLDataset1.Cells[1,i] := SQLDataset1.Fields[0].AsString;
    // Spalte Titel
    SQLDataset1.Cells[2,i] := SQLDataset1.Fields[1].AsString;
    SQLDataset1.Next;
    i := i + 1;
End;

```

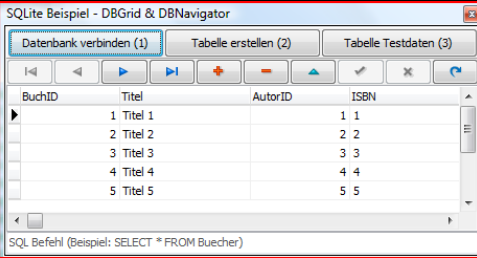


Abbildung 14

SELECT Spalten FROM Tabelle(n) WHERE Bedingung ORDER BY Sortierung GROUP BY Gruppierung HAVING Einschränkung

Bestimmte Datensätze lesen und als Ergebnis in einem Dataset zurückgeben.

SELECT Operator	Bedeutung
*	Alle Felder einer Tabelle anzeigen
DISTINCT	Felder kumuliert anzeigen
COUNT(*)	Anzahl Treffer zur optionalen Bedingung
GROUP BY	Ergebnis gruppieren
GROUP BY ... HAVING	Ergebnis gruppiert einschränken
ORDER BY ... ASC, DESC	Ergebnis auf- oder absteigend sortieren
Aggregatfunktionen	MIN(Feld), MAX(Feld), AVG(Feld), SUM(Feld), COUNT(Feld)
WHERE Operator	Bedeutung
=	Gleich
<>	Ungleich
>	Größer
<	Kleiner
>=	Größer Gleich
<=	Kleiner gleich
BETWEEN	Zwischen einem numerischen Bereich
LIKE	Ungefähre Übereinstimmung Platzhalter % (beliebig viele Zeichen), _ (ein Zeichen)
IN	einem Wertesatz
IS NULL	Leere Einträge abfragen
IS NOT NULL	Keine leeren Einträge abfragen
AND, OR	Verknüpfung logischer Operationen

BEISPIELE

Anhand TSQLDataset	SQLDataset.CommandText :=
Alle Datensätze der Tabelle Buecher lesen	'SELECT * FROM Buecher';
Nur Buch mit ISBN 1 selektieren	'SELECT * FROM Buecher WHERE ISBN = "ISBN 1" ';
Alle Datensätze sortiert nach Titel absteigend DESC; oder aufsteigend ASC	'SELECT * FROM Buecher ORDER BY Titel DESC';
Alle Datensätze mit AutorID und Autorname (SELECT auf mehrere Tabellen (JOIN))	'SELECT Buecher.BuchID,Buecher.Titel,Buecher.AutorID, Autoren.Name,Buecher.ISBN FROM Autoren, Buecher WHERE Autoren.AutorID=Buecher.AutorID';
SQL-Befehl ausführen. Das Ergebnis wird im SQLDataset gespeichert.	SQLDataset.Open;

UPDATE Tabelle SET Spalte1=Wert, SpalteN=Wert WHERE Spalte=Bedingung

Beispiel: Tabelle Buecher aktualisieren.

Aufgabe: Titel 2 in Buchtitel 2 ändern

```
SQLConnection.ExecuteDirect('UPDATE Buecher SET Titel="Buchtitel 2", isbn="ISBN 2-2" WHERE Titel="Titel 2";');
```

DELETE FROM TABLE WHERE Spalte=Bedingung

Beispiel: Alle Datensätze aus der Tabelle Buecher löschen.

```
SQLConnection.ExecuteDirect('DELETE FROM Buecher;');
```

Beispiel: Datensatz mit der BuchID=1 aus der Tabelle Buecher löschen.

```
SQLConnection.ExecuteDirect('DELETE FROM Buecher WHERE BuchID=1;');
```

DROP TABLE Tabellenname

Beispiel: Tabelle Buecher aus der Datenbank löschen.

```
SQLConnection.ExecuteDirect('DROP TABLE IF EXISTS Buecher;');
```

CREATE INDEX Indexname ON Tabelle(Spalte)

Beispiel: Auf Spalte ISBN einen eindeutigen Index erstellen.

```
SQLConnection.ExecuteDirect('CREATE UNIQUE INDEX idx_isbn ON Buecher(ISBN);');
```

HINWEIS

- SQLite verwendet den Index automatisch.

DROP INDEX Indexname

Beispiel: Index idx_isbn löschen.

```
SQLConnection.ExecuteDirect('DROP INDEX IF EXISTS idx_isbn;');
```

VIEWS

Views sind Tabellen, die ein bestimmtes Ergebnis zeigen.

Auf Views können SQL `SELECT` Befehle ausgeführt werden.

CREATE VIEW Name AS SELECT Befehl

Verschiedene Beispiele (verwendet Var `LSQL`: String für den SQL-Befehl).

View erstellen auf alle Buecher:

```
LSQL := 'CREATE VIEW AlleBuecher AS SELECT * FROM Buecher;';  
SQLConnection.Execute(LSQL, NIL);
```

Alle Buecher zeigen:

```
LSQL := 'SELECT * FROM AlleBuecher;';  
SQLConnection.Execute(LSQL, NIL, FDLSQLResults);
```

View erstellen auf alle Buecher die eine „1“ im Titel haben:

```
LSQL := 'CREATE VIEW TitellBuecher AS SELECT * FROM Buecher WHERE Titel LIKE "%1";'  
SQLConnection.Execute(LSQL, NIL);  
// Alle Datensätze View TitellBuecher zeigen  
LSQL := 'SELECT * FROM TitellBuecher;';  
SQLConnection.Execute(LSQL, NIL, FDLSQLResults);
```

DROP VIEW Viewname

View löschen:

```
LSQL := 'DROP VIEW IF EXISTS AlleBuecher';  
SQLConnection.Execute(LSQL, NIL);
```

TRIGGER

Ein Trigger ist ein Ereignis, das ausgeführt wird, wenn in der Datenbank eine bestimmte Aktion (wie zum Beispiel SQL-Befehl Insert, Update, Delete) durchgeführt wird.

CREATE TRIGGER Name AFTER Aktion ON Tabelle BEGIN Ereignis; END

BEISPIEL AUFGABE

Alle Änderungen der Tabelle Buecher mit den SQL-Befehlen INSERT, UPDATE oder DELETE, in eine Tabelle Log schreiben.

Das Änderungsdatum und das Buch, das geändert wurde, sollen gespeichert („logged“) werden.

Tabelle Log erstellen

SQL-Befehle als Konstanten definieren:

```
Const
  CSQLDROPTABLELOG = 'DROP TABLE IF EXISTS Log;';
  CSQLCREATETABLELOG = 'CREATE TABLE Log (' +
    'LogID INTEGER PRIMARY KEY AUTOINCREMENT NOT NULL UNIQUE, Datum VARCHAR(20), Info' +
    ' VARCHAR(255));';
```

In eine Prozedur, z.B. FormCreate die Tabelle Log neu erstellen:

```
SQLConnection.ExecuteDirect(CSQLDROPTABLELOG);
SQLConnection.ExecuteDirect(CSQLCREATETABLELOG);
```

Trigger erstellen

SQL-Befehle als Konstanten definieren:

```
Const
CR = #13#10;
CSQLDROPTRIGGERBI =
'DROP TRIGGER IF EXISTS Buecher_I;';
CSQLCREATETRIGGERBI =
'CREATE TRIGGER Buecher_I AFTER INSERT ON Buecher ' + CR +
'BEGIN' + CR +
' INSERT INTO Log(Datum, Info) ' + CR +
' VALUES (DATETIME('now'), 'Neu: ' || new.Titel);' + CR +
'END;';
CSQLDROPTRIGGERBU =
'DROP TRIGGER IF EXISTS Buecher_U;';
CSQLCREATETRIGGERBU =
'CREATE TRIGGER Buecher_U AFTER UPDATE ON Buecher ' + CR +
'BEGIN' + CR +
' INSERT INTO Log(Datum, Info) ' + CR +
' VALUES (DATETIME('now'), 'Geändert: ' || old.Titel || ' auf ' || new.Titel);'+
CR +
'END;';
CSQLDROPTRIGGERBD =
'DROP TRIGGER IF EXISTS Buecher_D;';
CSQLCREATETRIGGERBD =
'CREATE TRIGGER Buecher_D AFTER DELETE ON Buecher ' + CR +
'BEGIN' + CR +
' INSERT INTO Log(Datum, Info) ' + CR +
' VALUES (DATETIME('now'), 'Gelöscht: ' || old.Titel);' + CR +
'END;';
```

In eine Prozedur, z.B. FormCreate die Trigger neu erstellen:

```
SQLConnection.ExecuteDirect(CSQLDROPTRIGGERBI);
SQLConnection.ExecuteDirect(CSQLCREATETRIGGERBI);
    usw. für TRIGGERBU und TRIGGERBD.
```

Änderungen der Tabelle „Log“ anzeigen

SELECT * FROM Log;
Es wurde Titel hinzugefügt.

LogID	Datum	Info
1	2013-03-17 10:24:20	Neu: Schwere Wetter
2	2013-03-17 10:24:20	Neu: Nebelfront
3	2013-03-17 10:24:20	Neu: Flutgrab
4	2013-03-17 10:24:20	Neu: Knochenwald
5	2013-03-17 10:24:20	Neu: Verachtung
6	2013-03-17 10:24:20	Neu: Erlösinn

Abbildung 15

Alle Einträge der Tabelle „Log“ löschen

```
DELETE FROM Log;
```

PRAGMA

PRAGMA sind spezielle SQLite Befehle, um interne Informationen aus der Datenbank zu ermitteln. PRAGMA Befehle werden wie SQL-Befehle ausgeführt.

Das Ergebnis wird wiederum im Dataset gespeichert (siehe unter „SQL SELECT ausführen“).

Beispiel PRAGMA Befehl ausführen und Ergebnis im TMemo zeigen:

```
...
SQLConnection.Execute(LSQL, NIL, FDLSQLResults);
FDLSQLResults.First;
While Not FDLSQLResults.EOF Do Begin
    currentLine := '';
    For i := 0 to FDLSQLResults.FieldCount - 1 do
        currentLine := currentLine + ' ' + FDLSQLResults.Fields[i].AsString;
    // Datensatz hinzufügen
    memoSQLOutput.Lines.Add(currentLine);
    FDLSQLResults.Next;
End;
...
```

PRAGMA database_list

Liste der Datenbanken:

```
LSQL := 'PRAGMA database_list;';
SQLConnection.Execute(LSQL, NIL, FDLSQLResults);
```

ERGEBNIS

```
0 main E:\Daten\ropbooks\ropdelphisqlite\bookprojects\ropdelphisqlite\buecher.db
```

PRAGMA table_info (Tabellenname)

Tabellenstruktur ermitteln:

```
LSQL := 'PRAGMA table_info(Buecher);';
SQLConnection.Execute(LSQL, NIL, FDLSQLResults);
```

ERGEBNIS

Ausgebene wird folgende Feldinformation:

CID = SPALTENNUMMER

NAME = SPALTENNAME

TYPE = SPALTENTYP

NOTNULL = OB KEIN WERT ERLAUBT IST

DFLT_VALUE = VORGABEWERT

PK = OB SPALTE TEIL DES "PRIMARY KEY" IST

```
(cid, name, type, notnull, dflt_value, pk)
(Spaltennr, Spaltenname, Data Type, NOT NULL, Vorgabewert, Primary Key)
0 id INTEGER 1 1
1 titel TEXT 1 0
2 autor TEXT 0 0
3 isbn TEXT 0 0
```


PRAGMA index_list (Tabellenname)

Liste der Indices:

```
LSQL := 'PRAGMA index_list(Buecher);';  
SQLConnection.Execute(LSQL, NIL, FDLSQLResults);
```

ERGEBNIS

```
0 idx_isbn 1  
1 sqlite_autoindex_Books_1 1
```

PRAGMA index_info (Indexname)

Indexstruktur ermitteln:

```
LSQL := 'PRAGMA index_info(idx_isbn);';  
SQLConnection.Execute(LSQL, NIL, FDLSQLResults);
```

ERGEBNIS

```
0 3 isbn
```

PRAGMA auto_vacuum

Datenbank defragmentieren:

```
LSQL := 'PRAGMA auto_vacuum;';  
SQLConnection.Execute(LSQL, NIL, FDLSQLResults);
```

ERGEBNIS

```
Es wird kein Ergebnis zurück geliefert.
```

PRAGMA integrity_check

Datenbank Integrität prüfen:

```
LSQL := 'PRAGMA integrity_check;';  
SQLConnection.Execute(LSQL, NIL, FDLSQLResults);
```

ERGEBNIS

```
OK oder Hinweis.
```

SQLite SELECT Beispiele

SELECT ausführen

Wie schon erwähnt, liefert der SQL-Befehl **SELECT** ein Ergebnis in einem TDataSet.

Beispiel SQL-Befehl **SELECT und Ergebnis in einem TMemo „memoSQLOutput“ ausgeben:**

```
Var
  fSLNames: TStringList; // Liste Feldnamen Tabelle
  i: Integer;             // Laufvariable
  currentField: TField;   // Das aktuelle Tabellenfeld
  currentLine: string;    // Aktuelle Memozeile Feldergebnis
Begin
  memoSQLOutput.Lines.Add('Datensätze: ' + IntToStr(FDLSQLResults.RecordCount));
  fSLNames := TStringList.Create; // Liste der Feldnamen initialisieren
  FDLSQLResults.GetFieldNames(fSLNames); // Feldnamen lesen Ergebnisdataset
  FDLSQLResults.First; // Ergebnis Satz für Satz lesen
  While NOT FDLSQLResults.Eof Do Begin
    currentLine := '';
    For i := 0 To fSLNames.Count - 1 Do Begin
      currentField := FDLSQLResults.FieldByName(fSLNames[i]);
      currentLine := currentLine + ' ' + currentField.AsString;
    End;
    memoSQLOutput.Lines.Add(currentLine); // Datensatz anzeigen
    FDLSQLResults.Next;
  End;
  fSLNames.Free;
End;
```

WICHTIG

Es gibt Situationen, in denen nicht alle Informationen des **SELECT** Befehls im TDataSet gespeichert werden. Obwohl TDataSet Daten enthält, kann es zu Fehlermeldungen kommen, wie „Reader has no more rows“. Ein Beispiel ist die Anwendung von **TDataSet.RecordCount**. Um die Anzahl der Datensätze zu ermitteln, ist es sicherer, eine eigene Funktion zu definieren, anstatt RecordCount zu verwenden.

```
Function TfrmMain.DatasetRecordCount(DS: TDataSet): Integer;
Var i : Integer;
Begin
  Result := 0;
  If DS = NIL Then Exit;
  If DS.IsEmpty Then Exit;
  i := 0;
  DS.First;
  While Not DS.Eof Do Begin
    i := i + 1;
    DS.Next;
  End;
  Result := i;
End;
```

```
Aufruf: nCnt := DatasetRecordCount(FDSSQLResults);
```

SELECT Ergebnis im StringGrid darstellen

Beispiel Ergebnis SQL-Befehl SELECT in einem StringGrid ausgeben.

StringGrid Komponente Form hinzufügen:

sgSQLResults: TStringGrid;

Prozedur erstellen der das SQL-Ergebnis in einem StringGrid zeigt

```
Procedure TfrmMain.ActionSQLResultsToStringGridExecute(Sender: TObject);
Var
  nCol, nLin: Integer;
Begin
  If Not FDLSQLResults.IsEmpty Then Begin
    sgSQLResults.ColCount := FDLSQLResults.FieldCount + 1;
    sgSQLResults.RowCount := FDLSQLResults.RecordCount + 1;
    sgSQLResults.FixedCols := 1;
    sgSQLResults.FixedRows := 1;
    // Spaltenüberschriften setzen
    For nCol := 0 To FDLSQLResults.FieldCount - 1 Do
      sgSQLResults.Cells[nCol + 1, 0] := FDLSQLResults.Fields[nCol].FieldName;
    nLin := 0;
    // Daten im Stringgrid zeigen
    FDLSQLResults.First;
    While Not FDLSQLResults.Eof Do Begin
      For nCol := 0 To FDLSQLResults.FieldCount - 1 Do Begin
        sgSQLResults.Cells[nCol + 1, nLin + sgSQLResults.FixedRows] :=
          FDLSQLResults.Fields[nCol].AsString;
      End;
      FDLSQLResults.Next;
      Inc(nLin);
    End;
  End;
End;
```

SQL-Befehl SELECT ausführen und Ergebnis im StringGrid zeigen

```
Try
  SQLConnection.Execute('SELECT * FROM Buecher', NIL, FDLSQLResults);
  ActionSQLResultsToStringGridExecute(Sender);
Except On E: EDatabaseError Do ShowMessage(E.Message);
End;
```

SELECT Ergebnis TLabel und TEdit zuweisen

Das Ergebnis des SQL SELECT Befehls wird in einem TDataSet gespeichert.

Die Werte aus dem TDataSet können entsprechenden Komponenten wie TLabel, TEdit, TLabeledEdit usw. zugewiesen werden.

```
Var
    ledTitel: TLabeledEdit;
    lblTitel: TLabel;

// Tabelle Buecher, Spalte Titel Wert an TLabeledEdit und TLabel zuweisen
ledTitel.Text := FDLSQLResults.FieldByName('Titel').AsString;
lblTitel.Caption := FDLSQLResults.FieldByName('Titel').AsString;
```

SELECT Ergebnis TStringList und TListBox zuweisen

Das Ergebnis des SQL SELECT Befehls in einem TStringList speichern und anschließend einer TListBox zuweisen.

Stringlist erstellen:

```
Function BookTitellList(LSQL: String) : TStringList;
// Stringlist mit Buchtitel erstellen.
// SQL-Befehl „SELECT Titel FROM Buecher;“.
Var i: Integer;
    fSL: TStringList;
    DS: TDataSet;
Begin
    fSL := TStringList.Create;
    Try
        SQLConnection.Execute(LSQL, NIL, DS);
        DS.First;
        While Not DS.Eof Do Begin
            fSL.Add(DS.FieldByName('Titel').AsString);
            DS.Next;
        End;
    Except On E: Exception Do ShowMessage(E.Message);
    End;
    Result := fSL;
end;
```

StringList TListBox zuweisen:

```
// TListbox Komponente hinzufügen
Var
    lbCodeList: TListBox;
// Alle Buchtitel in der Listbox zeigen.
lbCodeList.Items := BookTitellList('SELECT Titel FROM Buecher;');
```

HINWEIS

- SELECT Befehl erweitern, z.B. mit WHERE ISBN = '123';

SELECT Ergebnis in HTML Tabelle ausgeben

Inhalt SQL-Tabelle in eine HTML Tabelle ausgeben und im Standardbrowser darstellen.

```

Procedure TfrmMain.ActionSQLSelectAllBooksToHTMLExecute(Sender: TObject);
Const
  CCRLF = #13#10;
  CHTMLCSS = '<style type="" + 'text/css' + ''>' + CCRLF +
    'table { width:100%; background-color:#D7E4F2; border:1px; } ' + CCRLF +
    'h1 { font:bold 2.0em Arial; color:#000080; background-color:#; border:1px; solid #000;
    vertical-align:top; overflow:hidden; }' + CCRLF +
    'th { font:bold 1.6em Arial; color:#000080; background-color:#F0F8FF; border:1px; solid
    #000; vertical-align:top; overflow:hidden; }' + CCRLF +
    'td { font:normal 1.0em Arial; color:#000000; background-color:#; border:1px; solid
    #000; vertical-align:top; overflow:hidden; }' + CCRLF +
    'div { font:normal 0.6em Arial; color:#000080; background-color:#; border:1px; solid
    #000; vertical-align:top; overflow:hidden; }' + CCRLF +
    '</style>';
  CHTMLBEGIN = '<HTML>' + CCRLF +
    '<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN">' + CCRLF +
    '<HEAD>' + CCRLF + CHTMLCSS + CCRLF +
    '</HEAD><BODY bgcolor=#FFFFFF><H1>Buchtitel</H1><TABLE>';
  CHTMLEND = '</TABLE><p><div>Erstellt mittels ropDelphiSQLite (c) 2013 Robert W.B. Linn,
  Pinnberg, Germany</div></p></BODY></HTML>';
Var DS: TDataSet;
    fSL: TStringList;
    sFile, sField: String;
    i: Integer;
Begin
  Try
    SQLConnection.Execute('SELECT * FROM Buecher', NIL, DS);
  Except On E: Exception Do Begin
    ShowMessage(E.Message); Exit;
  End;
End;
If DS.IsEmpty Then Exit;
sFile := ExtractFilePath(Application.ExeName) + 'booklist.htm';
fSL := TStringList.Create;
fSL.Add(CHTMLBEGIN);
fSL.Add('<TR>');
For i := 0 To FieldCount - 1 Do
  fSL.Add('<TH>' + DS.Fields[i].FieldName + '</TH>');
fSL.Add('</TR>');
DS.First;
While Not DS.Eof Do Begin
  fSL.Add('<TR>');
  For i := 0 To DS.FieldCount - 1 Do Begin
    sField := DS.Fields[i].AsString;
    If sField = '' Then sField := '&nbsp;';
    fSL.Add('<TD>' + sField + '</TD>');
  End;
  DS.Next;
  fSL.Add('</TR>');
End;
fSL.Add(CHTMLEND);
fSL.SaveToFile(sFile);
ShellExecute(Handle, NIL, PChar(sFile), NIL, NIL, SW_SHOW);
End;

```

SELECT auf mehrere Tabellen (JOIN)

BEISPIEL

Alle Autor Name (Tabelle Autoren) und Titel (Tabelle Buecher) sortiert nach Autor Name ausgeben.

SQL-Befehl SELECT und Ergebnis:

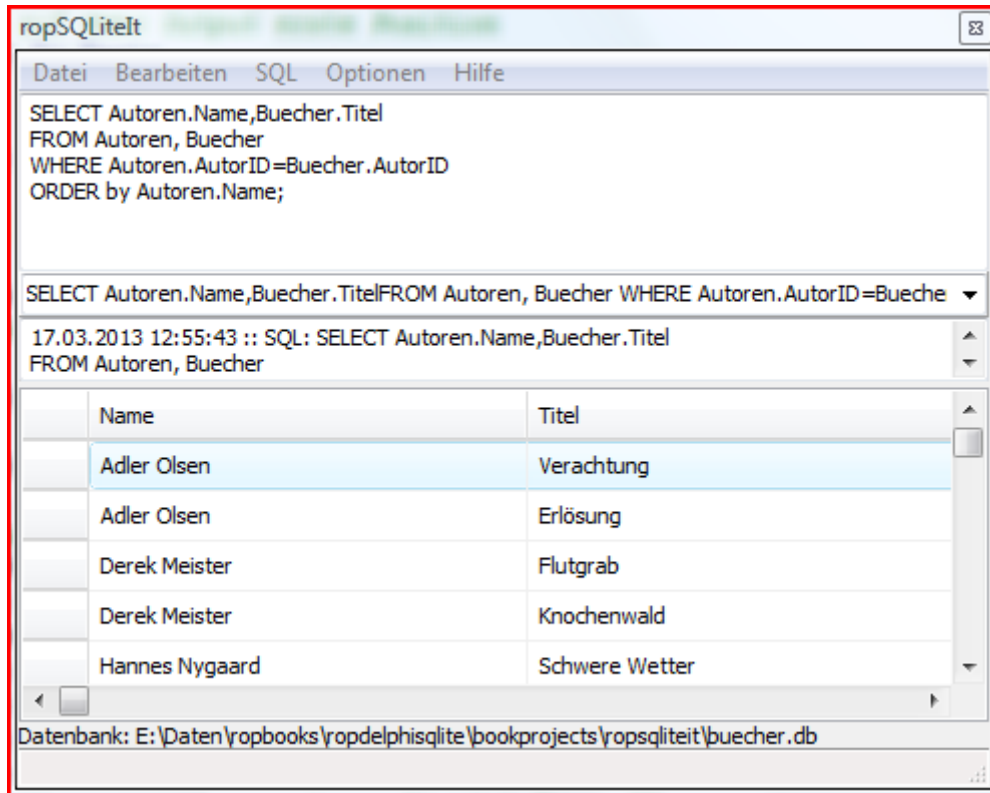


Abbildung 16

SELECT sqlite_master

Die spezielle SQLite Tabelle **sqlite_master** enthält alle Informationen über Tabellen, Indizes, Views, Trigger einer SQLite Datenbank.

Der SELECT Befehl verwenden um Informationen aus der Tabelle **sqlite_master** zu lesen.

Das Ergebnis sind die Spalten type (table, index, view, trigger), name, tbl_name, rootpage und sql.

BEISPIELE ANHAND DER DATENBANK BUECHER.DB

ALLE DATENSÄTZE DER TABELLE SQLITE_MASTER

SELECT * FROM sqlite_master;

type	name	tbl_name	rootpage	sql
index	sqlite_autoindex_Autoren_1	Autoren	6	
index	sqlite_autoindex_Buecher_1	Buecher	3	
index	sqlite_autoindex_Log_1	Log	8	
table	sqlite_sequence	sqlite_sequence	4	CREATE TABLE sqlite_sequence(name,seq)
table	Autoren	Autoren	5	CREATE TABLE Autoren (AutorID INTEGER PRIMARY KEY AUTOINCR...
table	Buecher	Buecher	2	CREATE TABLE Buecher (BuchID INTEGER PRIMARY KEY AUTOINCR...
table	Log	Log	7	CREATE TABLE Log (LogID INTEGER PRIMARY KEY AUTOINCREMENT...
trigger	Buecher_I	Buecher	0	CREATE TRIGGER Buecher_I AFTER INSERT ON Buecher
trigger	Buecher_U	Buecher	0	CREATE TRIGGER Buecher_U AFTER UPDATE ON Buecher
trigger	Buecher_D	Buecher	0	CREATE TRIGGER Buecher_D AFTER DELETE ON Buecher

Abbildung 17

ALLE TABELLENNAMEN DER TABELLE SQLITE_MASTER

SELECT NAME FROM SQLITE_MASTER WHERE TYPE="TABLE" ORDER BY NAME;

name
Autoren
Buecher
Log
sqlite_sequence

SELECT CAST (Ausdruck AS Datentyp [(Länge)])

CAST in SQLite konvertiert einen Ausdruck von einem Datentyp in einen anderen.

BEISPIELE

Die Tabelle **sqlite_master** enthält alle SQLite Datenbankinformationen, wie Tabellen, Indices, Views, Triggers.

Ausgabe Liste der Tabellentypen und Tabellennamen:

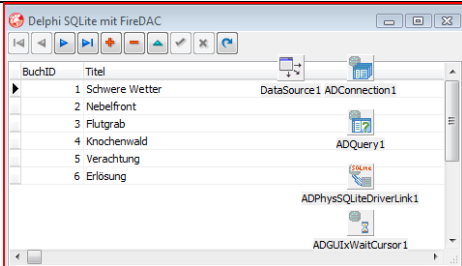
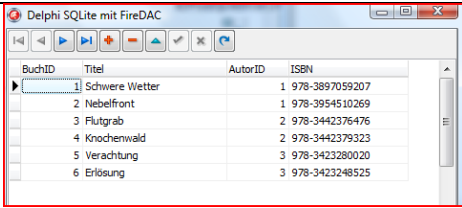
Ohne Cast	Mit Cast																																												
SELECT type, tbl_name FROM sqlite_master	SELECT type AS "Typ", (CAST(tbl_name AS VARCHAR(10))) AS "Tabellen" FROM sqlite_master																																												
<table> <tr><th>type</th><th>tbl_name</th></tr> <tr><td>table</td><td>sqlite_sequence</td></tr> <tr><td>table</td><td>Autoren</td></tr> <tr><td>index</td><td>Autoren</td></tr> <tr><td>table</td><td>Buecher</td></tr> <tr><td>index</td><td>Buecher</td></tr> <tr><td>table</td><td>Log</td></tr> <tr><td>index</td><td>Log</td></tr> <tr><td>trigger</td><td>Buecher</td></tr> <tr><td>trigger</td><td>Buecher</td></tr> <tr><td>trigger</td><td>Buecher</td></tr> </table> <p>Abbildung 18</p>	type	tbl_name	table	sqlite_sequence	table	Autoren	index	Autoren	table	Buecher	index	Buecher	table	Log	index	Log	trigger	Buecher	trigger	Buecher	trigger	Buecher	<table> <tr><th>Typ</th><th>Tabellen</th></tr> <tr><td>table</td><td>sqlite_sequence</td></tr> <tr><td>table</td><td>Autoren</td></tr> <tr><td>index</td><td>Autoren</td></tr> <tr><td>table</td><td>Buecher</td></tr> <tr><td>index</td><td>Buecher</td></tr> <tr><td>table</td><td>Log</td></tr> <tr><td>index</td><td>Log</td></tr> <tr><td>trigger</td><td>Buecher</td></tr> <tr><td>trigger</td><td>Buecher</td></tr> <tr><td>trigger</td><td>Buecher</td></tr> </table> <p>Abbildung 19</p>	Typ	Tabellen	table	sqlite_sequence	table	Autoren	index	Autoren	table	Buecher	index	Buecher	table	Log	index	Log	trigger	Buecher	trigger	Buecher	trigger	Buecher
type	tbl_name																																												
table	sqlite_sequence																																												
table	Autoren																																												
index	Autoren																																												
table	Buecher																																												
index	Buecher																																												
table	Log																																												
index	Log																																												
trigger	Buecher																																												
trigger	Buecher																																												
trigger	Buecher																																												
Typ	Tabellen																																												
table	sqlite_sequence																																												
table	Autoren																																												
index	Autoren																																												
table	Buecher																																												
index	Buecher																																												
table	Log																																												
index	Log																																												
trigger	Buecher																																												
trigger	Buecher																																												
trigger	Buecher																																												

SQLite und FireDAC Einstieg

Eine einfache Anwendung, Liste der Bücher aus der Beispieldatenbank Buecher.db, zeigt die Verwendung der FireDAC Komponenten.

HINWEIS

Eine detaillierte Beschreibung wie SQLite unter FireDAC verwendet wird, ist in der FireDAC zu finden unter „Using SQLite with FireDAC“ (siehe Delphi DIE Menü FireDAC > Help).

Aktivität	Beschreibung
Delphi Projekt erstellen oder öffnen, Form wählen	Vgl. frmMain HINWEIS Sicherstellen das unter Project > Options > Delphi Compiler > Unit output directory ein Verzeichnis angegeben ist. Vgl. .\\$(Platform)\\$(Config) oder .\
Komponenten hinzufügen	
TADConnection	<pre> Params.Strings = ('DriverID=SQLite' 'Database=buecher.db' 'OpenMode=ReadWrite' 'SharedCache=false' 'LockingMode=Normal') FetchOptions.AssignedValues = [evMode] UpdateOptions.AssignedValues = [uvUpdateMode] UpdateOptions.UpdateMode = upWhereAll Connected = True LoginPrompt = False </pre>
TADQuery	<pre> Active = True Connection = ADConnection1 SQL.Strings = ('SELECT * FROM Buecher') </pre>
ADPhysSQLiteDriverLink ADGUIxWaitCursor	Keine Anpassungen notwendig.
TDatasource	DataSet = ADQuery1
TDBGrid	DataSource = DataSource1
TDBNavigator	DataSource = DataSource1
Entwurf	 <p>Abbildung 20</p>
Anwendungen ausführen	 <p>Abbildung 21</p>

SQLite spezielle Anwendungsbeispiele

SQLite Datensätze im DBGRID

Download Delphi XE3 Quellcode [ropDelphiSQLiteDBGrid](#)

Um eine SQLite Datenbank mit einem DBGrid zu verbinden, werden verschiedene Komponenten benötigt:

```
TSQLConnection (dbExpress) -> TSQLDataSet (dbExpress) -> TDataSetProvider (Data Access) -> TClientDataSet (Data Access) -> TDataSource (Data Access) -> TDBGrid (Data Controls)
```

Siehe grafische Darstellung (startet mit SQLConnection1 ... DataSource1)

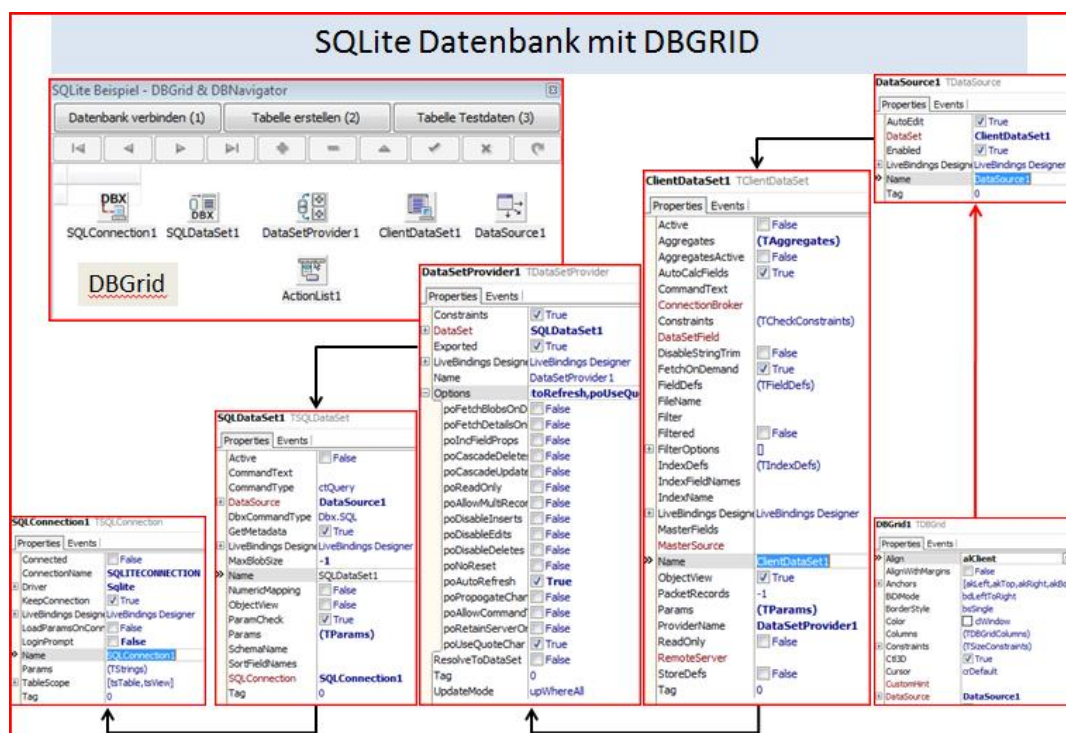


Abbildung 22

HINWEISE

- DBGrid1 ist mit DataSource1 verbunden.
- DBNavigator1 ist auch mit DataSource1 verbunden.
- Änderungen können im DBGrid1 direkt oder mittels DBNavigator1 vorgenommen werden.
- Wenn die Anwendungen verlassen werden, müssen die Daten aus dem DBGrid1 in die SQLite Datenbank gespeichert werden. Hierzu wird ClientDataSet1.ApplyUpdates verwendet.

AUSSCHNITTE AUS EINER BEISPIELANWENDUNG

Verwendet wird die SQLite Datenbank **BUECHER.DB** mit den Tabellen **Buecher** und **Autoren**.

Verbindung mit der SQLite Datenbank herstellen:

```

Procedure TfrmMain.ActionDatabaseConnectExecute(Sender: TObject);
Begin
  Try
    SQLDataset1.Close;
    SQLConnection1.Close;
    SQLConnection1.Params.Values['Database'] := 'buecher.db'; //ggf. Pfadangabe
    SQLConnection1.Params.Values['FailIfMissing'] := 'False';
    SQLConnection1.Open;
  Except On E: EDatabaseError Do ShowMessage(E.Message);
  End;
End;

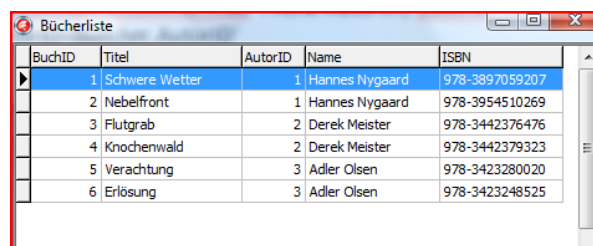
```

Daten selektieren und im DBGrid darstellen:

```

Procedure TfrmMain.ActionTableGetDataExecute(Sender: TObject);
// Daten aus der SQLite Tabelle Buecher und Autoren lesen und im DBGrid anzeigen
// Verwendet wird ein SQL Select JOIN Befehl.
Begin
  Try
    ClientDataSet1.Active := False;
    SQLDataset1.CommandText := 'SELECT Buecher.BuchID,Buecher.Titel,Buecher.AutorID,
Autoren.Name,Buecher.ISBN FROM Autoren, Buecher WHERE Autoren.AutorID=Buecher.AutorID;';
    SQLDataset1.Open;
    // Clientdataset aktivieren um die Daten im DBGrid anzuzeigen
    ClientDataSet1.Active := True;
  Except On E: EDatabaseError Do ShowMessage(E.Message);
  End;
End;

```

ERGEBNIS


BuchID	Titel	AutorID	Name	ISBN
1	Schwere Wetter	1	Hannes Nygaard	978-3897059207
2	Nebelfront	1	Hannes Nygaard	978-3954510269
3	Flutgrab	2	Derek Meister	978-3442376476
4	Knochenwald	2	Derek Meister	978-3442379323
5	Verachtung	3	Adler Olsen	978-3423280020
6	Erlösung	3	Adler Olsen	978-3423248525

Abbildung 23

HINWEISE

- Bevor der SQLDataset1.CommandText ausgeführt wird, muss zuerst das SQLDataSet deaktiviert werden.
- Im DBGrid wird das Ergebnis des SQL-Befehls `SELECT` gezeigt.
Es können auch selektive Ergebnisse als SQLDataset1.CommandText gezeigt werden.
Vgl.
'SELECT * FROM Buecher WHERE BuchID < 3;'
oder
'SELECT * FROM Buecher ORDER BY ISBN DESC';
oder JOIN
'SELECT Autoren.Name,Buecher.Titel FROM Autoren, Buecher WHERE
Autoren.AutorID=Buecher.AutorID;'

Daten in einen DBGrid mittels DBNavigator einfügen – spezieller Fall AutoInc Feld

Wird ein AutoInc Feld benutzt, dann muss dieses Feld mit einem Wert belegt werden, bevor der Datensatz im CleintDataSet gespeichert wird. Unter Verwendung eines DBNavigators, macht SQLite das nicht automatisch.

BEISPIEL TABELLE BUECHER, FELD BUCHID (AUTOINC).

Definiert wird eine Prozedur ClientDataSetBeforePost, die eine Funktion GetNextBuchID verwendet.

```
Procedure TfrmMain.ClientDataSet1BeforePost(DataSet: TDataSet);
Begin
    // Feld BuchID auf NULL prüfen und den nächst höheren Wert zuweisen.
    If TClientDataSet(DataSet).FieldByName('BuchID').IsNull Then
        TClientDataSet(DataSet).FieldByName('BuchID').Value := GetNextBuchID;
End;
```

```
Function TfrmMain.GetNextBuchID: Integer;
// Nächst höheren Wert für Feld BuchID definieren
Begin
    SQLDataset1.Close;
    SQLDataset1.CommandText := 'SELECT Max(BuchID) FROM Buecher;';
    SQLDataset1.Open;
    Result := SQLDataset1.Fields[0].AsInteger + 1;
    SQLDataset1.Close;
End;
```

Daten aus dem ClientDataSet in der SQLite Datenbank speichern

Im DBGrid können Daten bearbeitet werden. Da diese im Speicher verbleiben, müssen Änderungen vom ClientDataSet auch in der SQLite Datenbank gespeichert werden.

Verwende hierzu ClientDataSet.ApplyUpdates.

```
Procedure TfrmMain.ClientDataSet1BeforeRefresh(DataSet: TDataSet);
Begin
    If DataSet = NIL Then Abort;
    If TClientDataSet(DataSet).Active = False Then Abort;
    If TClientDataSet(DataSet).ChangeCount > 0 Then begin
        TClientDataSet(DataSet).ApplyUpdates(-1);
    end;
End;
```

HINWEIS

Um sicherzustellen, dass Änderungen auch beim Verlassen der Anwendung in der SQLite Datenbank gespeichert werden, ClientDataSet1BeforeRefresh() in FormClose aufrufen.

```
Procedure TfrmMain.FormClose(Sender: TObject; var Action: TCloseAction);
Begin
    ClientDataSet1BeforeRefresh(ClientDataSet1);
End;
```

SQLite BLOBs verwenden

Download Delphi XE3 Quellcode [ropDelphiSQLiteBlob](#).

Tabelle mit BLOB Spalte definieren:

```
Try
  SQLConnection.ExecuteDirect('CREATE TABLE images (' +
    'id INTEGER PRIMARY KEY AUTOINCREMENT NOT NULL UNIQUE,' +
    'title TEXT NOT NULL,' +
    'image BLOB NOT NULL)');
Except On E: EDatabaseError Do ShowMessage(E.Message);
End;
```

BLOB in Tabelle images Feld image laden:

```
Procedure TfrmMain.InsertPicture(Sender: TObject; sTitle, sImagePath: String);
// Titel und Image in der SQL-Tabelle images laden
Var
  streamImage: TMemoryStream;      // Stream um Datei Image zu speichern
  LTransaction: TDBXTransaction;    // SQLConnection Transaktion
  LDBXCmd: TSQLQuery;              // SQL-Query
  LParam: TParam;                  // Parameter für SQL-Query
  LSQL: String;                    // SQL-Befehl
begin
  // SQL-Befehl INSERT zusammenstellen - achte auf "value ?" für image
  LSQL := 'INSERT INTO images(title, image) VALUES(' + sTitle + ', ?)';
  // Stream erstellen - LoadFromFile kann noch erweitert werden mit z.B.
  // If FileExists...
  streamImage := TMemoryStream.Create;
  streamImage.LoadFromFile(sImagePath);
  // Transaktion eröffnen
  LTransaction := SQLConnection.BeginTransaction;
  // Query Befehl zusammenstellen
  LDBXCmd := TSQLQuery.Create(SQLConnection);
  Try
    Try
      LDBXCmd.SQLConnection := SQLConnection;
      LDBXCmd.SQL.Text := LSQL;
      LParam := LDBXCmd.Params.CreateParam(ftBlob, 'Picture', ptInput);
      LParam.LoadFromStream(streamImage, ftBlob);
      LDBXCmd.ExecSQL;
    Except On E: Exception Do SQLConnection.RollbackFreeAndNil(LTransaction);
    End;
    SQLConnection.CommitFreeAndNil(LTransaction);
  Finally
    LDBXCmd.Free;
  End;
End;
```

Aufrufbeispiel Bitmap m.bmp aus dem Anwendungs-/Projektverzeichnis in der Tabelle laden

```
InsertPicture(Sender, 'M', ExtractFilePath(Application.ExeName) + 'm.bmp');
```

Inhalt BLOB in Formular TImage zeigen:

```

Procedure TfrmMain.LoadPicture(Sender: TObject; sTitle: String);
// Titel und Bild in Label und Image laden
Var
  LSQL: String;           // SQL-Befehl
  DS: TDataSet;           // Resultat SQL-Befehl Select
  BlobField: TField;      // Blobfeld benötigt für BlobStream
  BS: TStream;            // BS Blobstream
  BM: TBitmap;            // Bitmap im Image laden
Begin
  // Select Befehl nach Titel
  LSQL := 'SELECT * FROM images WHERE title = ' + sTitle + ' ';
  Try
    // SQL-Befehl ausführen und Ergebnis in DS speichern
    DS := NIL;
    SQLConnection.Execute(LSQL, NIL, DS);
    // Ergebnis lesen und Inhalt Label, BlobField, Bitmap und Image zuweisen
    If DS <> NIL Then Begin
      lblTitle.Caption := DS.FieldByName('title').AsString;
      Blobfield := DS.FieldByName('image');
      BS := DS.CreateBlobStream(BlobField, bmReadWrite);
      BM := TBitmap.Create;
      BM.LoadFromStream(BS);
      Image1.Picture.Bitmap := BM;
    End;
  Except On E: Exception Do ShowMessage(E.Message);
  End;
End;

```

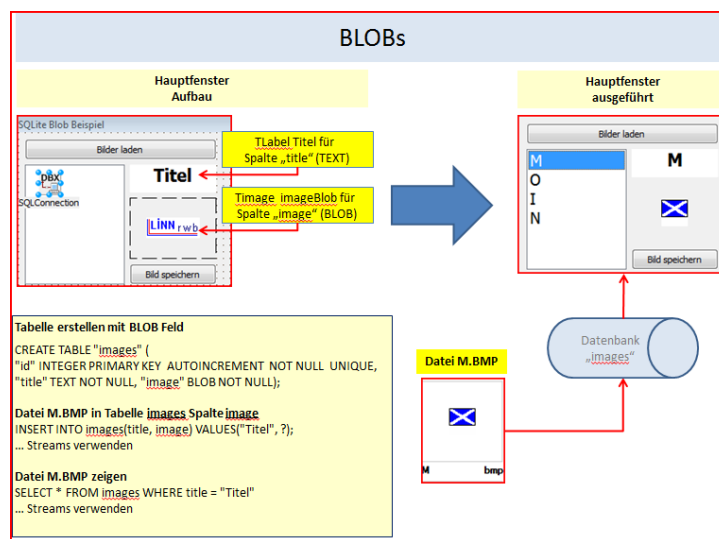


Abbildung 24

SQLite SQL-Befehle direkt ausführen

Download Delphi XE3 Quellcode [ropSQLiteIt](#)

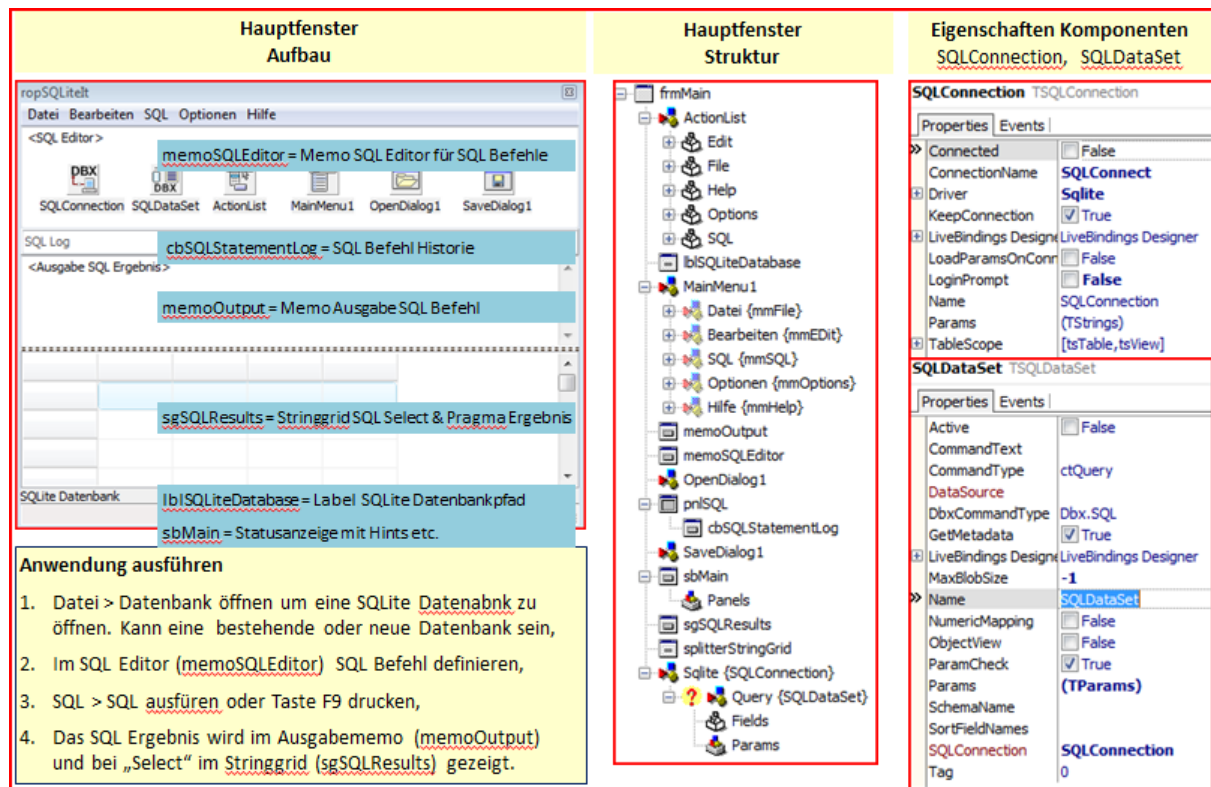


Abbildung 25

Beispiel wie ropSQLiteIt verwendet werden kann:

Eine Datenbank Logs.db mit Logging Tabelle Logs erstellen und verwenden.

1. Datei > Datenbank öffnen (Strg-O) > Dateiname logs.db

3. Tabelle Logs erstellen:

```
CREATE TABLE logs (id INTEGER PRIMARY KEY AUTOINCREMENT, logtype TEXT, logname TEXT, logmessage TEXT, logtime DATETIME);
```

4. Testdaten hinzufügen:

```
INSERT INTO logs (id,logtype,logname,logmessage,logtime) VALUES (NULL, 'TestLog1', 'TestEvent 1', 'Testaktivität 1', datetime('now'));
INSERT INTO logs (id,logtype,logname,logmessage,logtime) VALUES (NULL, 'TestLog2', 'TestEvent 2', 'Testaktivität 2', datetime('now'));
INSERT INTO logs (id,logtype,logname,logmessage,logtime) VALUES (NULL, 'TestLog1', 'TestEvent 3', 'Testaktivität 3', datetime('now'));
```

5. Testdaten zeigen:

```
SELECT * FROM logs;
SELECT * FROM logs WHERE strftime('%s', 'now')-strftime('%s', logtime) < 3600;
SELECT * FROM logs WHERE logtype = "TestLog1";
SELECT logtype, COUNT(logtype) FROM logs GROUP BY logtype;
```

6. Testdaten löschen:

```
DELETE FROM logs;
```

ERGEBNIS

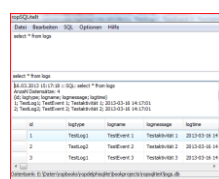


Abbildung 26

SQLite Tabelle für Einstellungen verwenden

Eine SQLite Tabelle kann auch verwendet werden, um Einstellungen einer Anwendung zu speichern oder zu lesen. Der Tabellenaufbau ist identisch mit der einer Ini-Datei: [SEKTION] Schlüssel=Wert.

SQL-Tabelle System definieren und erstellen:

```
CREATE TABLE System (
  ID INTEGER PRIMARY KEY AUTOINCREMENT NOT NULL UNIQUE,
  syssection TEXT,
  syskey TEXT,
  sysvalue TEXT);
```

Funktionen definieren um Werte zu speichern / lesen:

```
Function SQLSystemRead(sSection, sKey, sDefault : String): String;
Var LSQL: String;
    DS: TDataSet;
Begin
  Result := sDefault;
  Try
    LSQL := 'SELECT sysvalue FROM System WHERE ' +
            'syssection=' + ''' + sSection + ''' + ' AND ' +
            'syskey=' + ''' + sKey + ''' + ' ';
    SQLConnection.Execute(LSQL, NIL, DS);
    Result := DS.FieldByName('sysvalue').AsString;
    If Result = '' Then Result := sDefault;
  Except On E: EDatabaseError Do ShowMessage(E.Message);
  End;
End;
```

```
Function SQLSystemUpdate(sSection, sKey, sValue: String): Boolean;
Var LSQL: String;
    DS: TDataSet;
Begin
  Result := False;
  Try
    LSQL := 'SELECT sysvalue FROM System WHERE ' +
            'syssection=' + ''' + sSection + ''' + ' AND ' +
            'syskey=' + ''' + sKey + ''' + ' ';
    SQLConnection.Execute(LSQL, NIL, DS);
    If DS.FieldByName('sysvalue').AsString = '' Then Begin
      LSQL := 'INSERT INTO system (syssection, syskey, sysvalue) VALUES (' +
              ''' + sSection + ''' + ', ' +
              ''' + sKey + ''' + ', ' +
              ''' + sValue + ''' + ')';
      SQLConnection.Execute(LSQL, NIL);
      Result := True;
    End Else Begin
      LSQL := 'UPDATE system ' + 'SET ' +
              'syssection=' + ''' + sSection + ''' + ', ' +
              'syskey=' + ''' + sKey + ''' + ', ' +
              'sysvalue=' + ''' + sValue + ''' +
              ' WHERE ' +
              'syssection=' + ''' + sSection + ''' +
              ' AND ' +
              'syskey=' + ''' + sKey + ''' + ' ';
      SQLConnection.Execute(LSQL, NIL);
    End;
  End;
```

```
End;  
Except On E: EDatabaseError Do ShowMessage(E.Message);  
End;  
End;
```

Anwenden z.B. in FormCreate, FormClose

Procedure TfrmMain.FormCreate = Form Top Position lesen und setzen

```
Top := StrToInt(SQLSystemRead('MAINPOSITION', 'Top', '100'));
```

Procedure TfrmMain.FormClose = Form Top Position speichern

```
SQLSystemUpdate('MAINPOSITION', 'Top', IntToStr(Top));
```


Anhang

Fehlermeldungen und Abhilfe (unvollständige Liste!)

Eine unvollständige Liste von SQLite Fehlermeldungen (Englisch/Deutsch) mit Ursache und Abhilfe.

HINWEIS

In der SQLite Quellcode Datei sqlite3.c sind Fehlercodes und Hinweise kurz beschrieben.
Vgl.

```
#define SQLITE_OK          0   /* Successful result */
/* beginning-of-error-codes */
#define SQLITE_ERROR       1   /* SQL error or missing database */
...
```

Meldung	Ursache	Abhilfe
Sqlite3.dll not found	Die notwendige DLL Datei sqlite3.dll konnte nicht im Pfad gefunden werden.	Sqlite3.dll entweder im Projekt / Anwendungs-Verzeichnis oder Windows System32 Verzeichnis kopieren. Wenn sqlite3.dll nicht vorhanden ist, dann Download SQLite.org .
No such table: TABELLE	Ausführung SQL-Befehl: Die Tabelle TABELLE ist in der Datenbank nicht vorhanden.	Tabelle TABELLE erstellen oder SQL-Befehl ändern. Welche Tabellen in einer SQLite Datenbank enthalten sind, kann mittels PRAGMA table_info(TABALLE) ermittelt werden.
Operation not allowed on a unidirectional dataset	Meldung erscheint zum Beispiel wenn versucht wird ein SQLDataset mit einem DBGrid zu verbinden. Ein SQLDataset ist ein „unidirectionales“ Dataset, welches Datensätze nur in eine Richtung transportieren kann. Ein DBGrid kann Datensätze in beide Richtungen transportieren. Daher kann ein SQLDataset nicht direkt mit einem DBGrid verbunden werden.	Verwende DBGrid > DataSource > ClientDataset > DatasetProvider > SQLDataset > SQLConnection. Siehe SQLite Datenbank Datensätze im DBGrid (Direkt)
Unable to open database file	Datenbank wurde nicht gefunden.	SQLConnection Parameter auf False setzen: SQLConnection1.Params.Values['FailIfMissing'] := 'False'; oder Datenbankname prüfen.

Zusammenfassung einige SQL-Befehle (Beispiele)

Verwendete Variable:

```
// Ergebnis SQL-Befehl SELECT, PRAGMA in TDataSet speichern
Var FDLSQLResults: TDataSet;
// SQL-Befehl ausgeführt mittels SQLConnection.Execute
Var LSQL: String;
```

Tabelle Buecher erstellen, vorher löschen (falls vorhanden):

```
LSQL := 'DROP TABLE IF EXISTS Buecher';
SQLConnection.Execute(LSQL, NIL);
LSQL := 'CREATE TABLE Buecher (BuchID INTEGER PRIMARY KEY AUTOINCREMENT NOT NULL UNIQUE,
Titel VARCHAR(100) NOT NULL, AutorID INTEGER, ISBN VARCHAR(20));';
SQLConnection.Execute(LSQL, NIL);
```

Buecher hinzufügen:

```
LSQL := 'INSERT INTO Buecher (Titel, ISBN) VALUES ("Titel 1","ISBN1") ';
SQLConnection.Execute(LSQL, NIL);
SQLConnection.Execute(LSQL, NIL);
```

Buchtitel aktualisieren:

```
LSQL := 'UPDATE Buecher SET Titel="Buchtitel 2", ISBN="ISBN 2-2" WHERE Titel="Titel 2"';
SQLConnection.Execute(LSQL, NIL);
```

Alle Buecher zeigen. Ergebnis in TDataSet FDLSQLResults:

```
LSQL := 'SELECT * FROM Buecher;';
SQLConnection.Execute(LSQL, NIL, FDLSQLResults);
```

Und weitere....:

```
CREATE UNIQUE INDEX idx_isbn ON Buecher (isbn)
DROP INDEX idx_isbn
CREATE VIEW AlleBuecher AS SELECT * FROM Buecher
DROP VIEW AlleBuecher
```

Spezielle PRAGMA Befehle, wie Tabellenstruktur der Tabelle Buecher auflisten:

```
PRAGMA Table_Info(Buecher);
```

Referenz Beispieldatenbank Buecher.db SQL-Befehle

```
DROP TABLE IF EXISTS Autoren;
CREATE TABLE Autoren (AutorID INTEGER PRIMARY KEY AUTOINCREMENT NOT NULL UNIQUE, Name
VARCHAR(40), Beschreibung VARCHAR(255), Homepage VARCHAR(100));

DROP TABLE IF EXISTS Buecher;
CREATE TABLE Buecher (BuchID INTEGER PRIMARY KEY AUTOINCREMENT NOT NULL UNIQUE, Titel
VARCHAR(100) NOT NULL, AutorID INTEGER, ISBN VARCHAR(20));

DROP TABLE IF EXISTS Log;
CREATE TABLE Log (LogID INTEGER PRIMARY KEY AUTOINCREMENT NOT NULL UNIQUE, Datum
VARCHAR(20) NOT NULL, Info VARCHAR(255) NOT NULL);
DROP TRIGGER IF EXISTS Buecher_I;

CREATE TRIGGER Buecher_I AFTER INSERT ON Buecher
BEGIN
    INSERT INTO Log(Datum, Info)
    VALUES (DATETIME('now'), 'Neu: ' || new.Titel);
END;

DROP TRIGGER IF EXISTS Buecher_U;
CREATE TRIGGER Buecher_U AFTER UPDATE ON Buecher
BEGIN
    INSERT INTO Log(Datum, Info)
    VALUES (DATETIME('now'), 'Geändert: ' || old.Titel || ' auf ' || new.Titel);
END;

DROP TRIGGER IF EXISTS Buecher_D;
CREATE TRIGGER Buecher_D AFTER DELETE ON Buecher
BEGIN
    INSERT INTO Log(Datum, Info)
    VALUES (DATETIME('now'), 'Gelöscht: ' || old.Titel);
END;
```

```
INSERT INTO Autoren (AutorID,Name) VALUES (NULL,'Hannes Nygaard');
INSERT INTO Autoren (AutorID,Name) VALUES (NULL,'Derek Meister');
INSERT INTO Autoren (AutorID,Name) VALUES (NULL,'Adler Olsen');

INSERT INTO Buecher (BuchID,Titel,AutorID,ISBN) VALUES (NULL,'Schwere Wetter',1,'978-
3897059207');
INSERT INTO Buecher (BuchID,Titel,AutorID,ISBN) VALUES (NULL,'Nebelfront',1,'978-
3954510269');
INSERT INTO Buecher (BuchID,Titel,AutorID,ISBN) VALUES (NULL,'Flutgrab',2,'978-
3442376476');
INSERT INTO Buecher (BuchID,Titel,AutorID,ISBN) VALUES (NULL,'Knochenwald',2,'978-
3442379323');
INSERT INTO Buecher (BuchID,Titel,AutorID,ISBN) VALUES (NULL,'Verachtung',3,'978-
3423280020');
INSERT INTO Buecher (BuchID,Titel,AutorID,ISBN) VALUES (NULL,'Erlösung',3,'978-
3423248525');
```

```
SELECT Autoren.Name,Buecher.Titel FROM Autoren, Buecher WHERE
Autoren.AutorID=Buecher.AutorID;

SELECT Buecher.BuchID,Buecher.Titel,Buecher.AutorID, Autoren.Name,Buecher.ISBN FROM
Autoren, Buecher WHERE Autoren.AutorID = Buecher.AutorID
```