TurboDB 6

The Programmer's Guide to Powerful Database Applications

dataweb.
Smart Database Technologies

TurboDB 6

The desktop database engine to live with

by Peter Pohmann, dataweb

TurboDB is a full-featured multi-user database engine and a set of native components for accessing TurboDB database tables. TurboDB is available for Windows and .NET and supports Delphi and C++ Builder as well as Visual Studio.NET and all its programming languages.

TurboDB Components

Printed: November 2022 in Aicha, Germany

Table of Contents

1 TurboDB für VCL

1.1 Einführung

1.1.1 TurboDB Überblick

TurboDB umfasst eine vollwertige Multi-User Datenbank Engine und eine Reihe von Komponenten um aus einer Delphi/C++ Builder heraus auf TurboDB Tabellen zuzugreifen.

TurboDB für VCL

TurboDB Engine und TurboDB Komponenten sind 100% in Delphi geschrieben. TurboDB ähnelt im Gebrauch sehr stark den entsprechenden BDE Komponenten, die im Lieferumfang von Delphi Professional und Delphi Enterprise enthalten sind.

Im Vergleich zu den BDE Komponenten bietet TurboDB die folgenden Vorteile:

- Kleinere ausführbare Dateien
- Keine spezielle, aufwendige Installation und/oder Konfiguration nötig
- Volle Unterstützung von Unicode in Strings und Memos
- Tabellen können verschlüsselt werden
- Volltextindizierung für schnelle Stichwortsuche

Verglichen mit den Datenbank-Client-Komponenten für Interbase und MySql, hat TurboDB etliche Vorteile. TurboDB

- ist wesentlich leichter zu verwalten
- bietet Tabellenerzeugung und Änderung in der IDE
- beinhaltet eine Tabellen Komponente um auf Datenbanktabellen zugreifen zu können ohne SQL benutzen zu müssen
- ist weitgehend kompatibel zur BDE, eine Migration kann sehr schnell erfolgen
- bietet weit mehr Funktionalität über Methoden und Properties.

Voraussetzungen

Sie benötigen eines der folgenden Entwicklungs-Werkzeuge der Firma Borland/Embarcadero um mit TurboDB 6 arbeiten zu können: Delphi 6/7/2005/2006/2007/2009/2010/XE Professional oder höher, C++ Builder 6/2006/2007/2009/2010/XE Professional oder höher.

TurboDB (wie jede andere Datenbankzugriff-Technologie) arbeitet nicht mit einer Personal Edition oder Open Edition von Delphi/C++ Builder, da die Basis-Technologie von Borland/Embarcadero in diesen Produkten nicht enthalten ist.

Editionen

TurboDB ist in verschiedenen Versionen und Editionen verfügbar. Falls Sie weitere Fragen haben lesen Sie bitte FAQ oder bemühen Sie das TurboDB Team bei dataWeb. TurboDB benutzt die TurboDB Datenbank Engine, die im TurboDB Package enthalten ist.

TurboDB for .NET

dataweb bietet auch eine Datenbank Engine für .NET, die speziell in C# geschrieben wurde und keine zusätzlichen Rechte zur ausführung benötigt. Besuchen Sie unsere Homepage für weitere Informationen zu diesem Produkt.

1.1.2 Neue Features

TurboDB 6 bringt viel Neues, sowohl in der Datenbank Engine als auch in der VCL Komponenten Bibliothek. Die neuen Engine-Features umfassen Erweiterungen an TurboSQL und sind in der TurboDB Engine Dokumentation beschrieben. Die Änderungen an der Komponenten Bibliothek sind hier beschrieben:

- Unterstützung aller Sprachen durch Kollationen auf Tabellen- und Spalten-Ebene
- Backup der Datenbank bei laufendem Betrieb
- Wesentlich verbesserte Volltext-Indizierung und -Suche. In *TTdbTable.AddFulltextIndex2* kann eine Liste von Wort-Trennzeichen angegeben werden. Bei der Suche mit SQL kann eine Liste der zu durchsuchenden Spalten definiert werden.
- Auf Wunsch kann mit inkrementellen anstelle von statischen Filtern gearbeitet werden. Inkrementelle Filter sind deutlich schneller bei der Suche in großen Untermengen riesiger Datenmengen. Als Vorgabe bleiben die Vorteile statischer Filter erhalten.
- Unterstützung hierarchischer Suche (Drill-Down) und manueller modifizierter Filter durch ein neues Selektions-API in *TTdbDataSet*.
- *TTdbTable* verfügt nun über die Eigenschaft *IndexFieldNames* (wie *TTable*), die nach einer frei wählbaren Sequenz von Tabellenspalten sortiert. D.h, es muss kein Index für die Sortierung existieren.
- Die Größe des Speicher-Puffers (Cache) für Tabellendaten kann spezifiziert werden.
- Der VCL Auszählungstyp *ftWideMemo* wird unterstützt.
- Die Benutzeroberfläche des TurboDB Viewer wurde grundlegend überarbeitet, die Menüstruktur ist deutlich übersichtlicher geworden.
- TurboDB Viewer unterstützt die neue Bachup-Funktionalität und bietet eine neue Funktoin um die Kopie einer Tabelle in einer anderen Datenbank anzulegen.
- Der SQL Editor des TurboDB Viewer verfügt nun über Syntax-Highlighting und Code-Vervollständigung.
- TurboDB Viewer sortiert den Inhalt einer Tabelle durch Klick auf den Spaltenkopf.
- Und viele Features mehr auf Datenbankebene.

1.1.3 Upgrade einer Hautptversion

Um bestehende Projekte von Version 5 oder niedriger upzugraden sind einigen Anpassungen notwendig. Wenn Sie Ihre Anwendung zum ersten Mal mit den VCL Komponenten für TurboDB 5 übersetzen, müssen Sie alle Formulare in der IDE öffnen und den Ignorieren Knopf drücken, sooft eine Meldung mit Bezug auf TurboDB angezeigt wird. Nötig ist dies wegen weggefallener oder neu hinzugekommener Eigenschaften in der Komponenten Bibliothek, aber wenn Sie die Anweisungen in diesem Kapitel befolgen, wird Ihre Anwendung wie zuvor funktionieren.

Von TurboDB 5

TTdbDataSet.Filter und TTdbDataSet.Locate

Da TurboDB 6 echte Kollationen unterstützt und damit in der Tabellenspalte definiert, ob zwischen Groß- und Kleinschreibung unterschieden wird, sind die Werte *foCaseInsensitive* in der Eigenschaft *FilterOption* und *loCaseInsensitive* in *LocateOption* jetzt ohne Funktion. Falls diese Optionen bisher verwendet wurden, muss die Datenbanktabelle auf Level 6 gebracht und dabei die passende Kollation für die Tabelle oder Tabellenspalte definiert werden.

TTdbTable.LangDriver

Da TurboDB 6 echte Kollationen unterstützt , können Sprachtreiber nicht mehr verwendet werden. Zur formalen Kompatibilität existiert die Eigenschaft noch, wird aber nicht mehr genutzt. *TTdbTable* hat nun die neue Property *Collation*, um die Vorgabe-Kollation für seine textuellen Spalten zu definieren. Falls bisher ein Sprachtreiber verwendet wurde, ist die Tabelle auf Level 6 zu bringen und dabei eine passende Kollation zu definieren.

TField für Links, Relationem und Aufzählungen

In Delphi 2009 und höher werden Links, Relationen und Aufzählungen jetzt als Unicode Strings behandelt. Die entsprechenden Felder müssen daher vom Typ *TWideStringField* anstelle von

TStringField sein. Beim Konvertieren älterer Projekte müssen Felder dieser Typen gelöscht und erneut angelegt werden.

Von TurboDB 4

***Directory*-Eigenschaft**

Diese Eigenschaft war bereits als obsolet in TurboDB 4 gekennzeichnet und wurde nun endgültig entfernt. Verwenden Sie stattdessen die Eigenschaft Location.

***FilterType*-Eigenschaft**

Diese Eigenschaft wurde entfernt, da sie nicht mehr nötig ist. Mit TurboDB 5 können ist es möglich nach einer Bedingung und einem Schlüsselwort gleichzeitig zu suchen. Wenn Sie ein Formular, das eine *TTdbTable*-Komponente enthält, zum ersten Mal in der Delphi/C++ Builder IDE öffnen, klicken Sie in der Dialog Box, die Sie auf das Fehlen der Eigenschaft aufmerksam macht, einfach auf Ignorieren.

Tabellen Verschlüsselung

Tabellen verfügen nun über die Eigenschaften EncryptionMethod und Password anstelle von *Password* und *Key*. Falls Sie Ihre Tabelle schützen wollen, müssen Sie jetzt die *EncryptionMethod* angeben. Um mit TurboDB 4 kompatible zu sein, ist diese Eigenschaft bei Verwendung eines Schlüssels auf *temClassic* einzustellen oder auf *temProtection*, falls nur ein Passwort verwendet wurde. Falls Sie mit TurboDB 4 einen Schlüssel verwendet haben, lesen Sie bitte den Abschnitt über die Eigenschaft *Password*, um zu erfahren, wie Sie das anpassen können. Die Eigenschaft *Password* ist nun vom Typ *WideString* anstelle von *AnsiString* in TurboDB 4.

In Bezug dazu steht eine Änderung beim Ereignis OnPassword. Da *Key* in TurboDB 5 nicht mehr verwendet wird, wurde dieser Parameter aus der Signatur des Ereignisses entfernt.

Volltext-Indexe

Sie können in Ihrer VCL-Anwendung den bestehenden Code zur Volltextsuche weiter verwenden. Wenn Sie aber auf den neuen Tabellen-Level 4 upgraden wollen oder wenn Sie von den neuen Volltext-Index Features wie die deutlich verbesserte Performanz, Ranking und Wartung profitieren wollen, müssen Sie Ihren Programmcode etwas ändern, im Allgemeinen wird er aber einfacher werden.

Mit der neuen Volltextsuche ist es nicht mehr erforderlich die Tabelle mit den Schlüsselwörtern über eine Relation zu verknüpfen. Der Wortfilter funktioniert noch genauso, anstelle von *AddFulltextIndex* ist lediglich *AddFulltextIndex2* aufzurufen. Außerdem gibt es eine zusätzliche Methode UpdateFulltextIndex, die nur den Namen des Volltext-Index als Parameter benötigt.

Upgrade-Themen auf Datenbank Level

Weitere Betrachtungen speziell zu SQL Statements.

1.1.4 Upgrade einer Unterversion

Um sicherzugehen, dass eine bestehende Anwendung mit der neuen Version erstellt wird, sollten nach dem Upgrade einer TurboDB-Unterversion immer folgende Schritte vollzogen werden.

1. Aktivieren der Komponenten:
   Falls es noch nicht während der Installation von TurboDB gemacht wurde, müssen die Komponenten aktiviert werden. Das dazu nötige Werkzeug, den Aktivator, finden Sie im TurboDB Installationsverzeichnis oder einfach im Windows Startmenü. Während der Aktivierung muss die Delphi/C++Builder IDE geschlossen sein. Die Aktivierung kann überprüft werden, indem im lokalen Menü einer beliebigen TurboDB -Komponente der *About TurboDB* Dialog geöffnet wird. Hier ist die aktuelle Lizenz-Information angezeigt.
2. Überprüfen der Suchpfade
   Delphi/C++ Optionen
   Die Biblithekspfade in Delphi/C++ sollten den vollständigen Pfad zu den installierten TurboDB Komponenten oder die Umgebungsvariable *$(TurboDB6)* enthalten. Falls *$(TurboDB6)* verwendet wird, muss in den Delphi Umgebungsvariablen *$(TurboDB6)* den vollständigen Pfad zu den installierten TurboDB Komponenten aufweisen. Verweise auf

ältere TurboDB-Versionen sind zu entfernen.
Projekt Optionen
In den Projekt-Optionen dürfen die Suchpfade nicht von den IDE Vorgabewerten abweichen. Falls der TurboDB Pfad nicht in den Vorgabewerten enthalten ist, muss er hinzugefügt werden. Verweise auf ältere TurboDB-Versionen sind zu entfernen.

3. Beachten der Versionsänderungen:
   Lesen Sie die *readme* Datei. Wenden Sie die dort aufgelisteten Änderungen auf Ihre Projekte und Daten an.
4. Erzeugen der Projekte:
   Um sicherzugehen, dass die neuen Komponenten auch wirklich verwendet werden, müssen die Projekte neu erzeugt werden (Menüpunkt *Erzeugen*,***Compilieren* ist nicht ausreichend**).

1.1.5 Erste Schritte

Nach der in readme.txt beschriebenen Installation verfügt die Komponentenpalette Ihrer Delphi oder C++Builder IDE über eine zusätzliche Seite die TurboDB heißt. Die Online Hilfe wurde um das Buch TurboDB ergänzt.

Zu Beginn sollten Sie vielleicht einen Blick auf die Demo Anwendungen werfen, die in TurboDB enthalten sind. Sie finden die Projekte im samples Verzeichnis der TurboDB Installation. Laden Sie beispielsweise einfach das Projekt tdbdemo.dpr in die IDE und starten sie es. Eventuell müssen Sie die Datenbanknamen der TurboDB Tabellen im Objektinspektor anpassen.

Wenn Sie mit Ihrer eigenen Anwendungen beginnen, müssen Sie die passenden Datenbanktabellen erzeugen. Das TurboDB Paket beinhaltet zu diesem Zweck das visuelle Werkzeug TurboDB Viewer. Es befindet sich im *windows\bin* Verzeichnis Ihrer TurboDB Installation. Es ist auch möglich die Tabellen direkt in der IDE zu erstellen.

Um ein neues Projekt das TurboDB verwendet zu erstellen müssen Sie:

1. In der IDE ein neues Projekt erzeugen.
2. Für jede benötige Tabelle fügen Sie eine *TTdbTable* Komponente zu Ihrem Formular oder Datenmodul hinzu.
3. Öffnen Sie das Komponenten Menü jeder *TTdbTable* Komponente und wählen Sie *Properties* um die Tabellenstruktur anzulegen.
4. Setzen Sie die *Active* Eigenschaft der *TTdbTable* Komponente auf True, um die Tabelle zu öffnen.
5. Vergewissern Sie sich, dass das windows\delphiX, windows\cbX oder linux/kylixX Unterverzeichnis Ihrer TurboDB Installation im Delphi oder Kylix Suchpfad enthalten ist. (Ersetzen Sie das X durch die Version Ihres Delphi, oder C++ Builder.)

Verwenden Sie TurboDB wie Sie es von den BDE Datenbank Komponenten gewohnt sind. Diese Dokumentation beschreibt jede Methode und jedes Eigenschaft das die TurboDB Komponenten zu bieten haben.

1.1.6 Installation der VCL Edition

1.1.6.1 Windows Installation

Das # Zeichen steht als Platzhalter für die interne Versionsnummer von Rad Studio, Delphi bzw. C++Builder. Mit der Liste unten können Sie die richtige Nummer für Ihren Compiler bestimmen.

1. Beenden Sie alle laufenden Instanzen von Rad Studio, Delphi or C++ Builder.
2. Starten Sie TurboDBVCL6.msi. Das Installationsprogramm wird Altversionen von TurboDB deinstallieren, das Komponeneten-Package installieren und die TurboDB Palette in die IDE des Compilers integrieren. Falls die automatische Deinstallation der Altversion fehlschlägt,

deinstallieren Sie diese bitte manuell aus der Windows Systemsteuerung.

3. Starten Sie Delphi. Sie werden eine neue Komponentenpalette mit Namen TurboDB finden, die die neuen Komponenten enthält. Falls Sie keine TurboDB-Komponentenpalette sehen, so öffnen Sie *Komponente/Packages installieren*. Aktivieren Sie dort den Eintrag "dataweb TurboDB 6 VCL" oder benutzen Sie den "Hinzufügen..."-Button um das TurboDB 6 Design Time Package "dclturbodb6d##.bpl" zu installieren.
4. Wählen Sie *Tools* > *Umgebungsoptionen* > *Bibliothek* und fügen Sie den TurboDB Komponenten Pfad passend zu Ihrem Compiler dem Bibliothekspfad hinzu (Falls der Pfad noch nicht angegeben ist).
5. Das Installationsprogramm integriert die TurboDB Dokumentation direkt in das Hilfe System Ihres Compilers. Falls dies nicht erfolgreich war, können Sie auf die Hilfe Datei "TurboDB6.chm" im Unterverzeichnis "Documentation" der TurboDB Installation zugreifen.
6. Alternativ können Sie die Online Dokumentation auch direkt auf unserer Homepage lesen.
7. Falls Sie Probleme mit der Installation haben, wenden Sie sich bitte an das dataweb Support Team.

Compiler Versionen and Pfade

Interne Versions-Nummer	Compiler	Pfad zu TurboDB 6 Komponenten	Platform
6	Delphi / C++ Builder 6	<TurboDB 6 Installation>\Lib\Delphi6	Win32
7	Delphi 7	<TurboDB 6 Installation>\Lib\Delphi7	Win32
9	Delphi 2005	<TurboDB 6 Installation>\Lib\Delphi9	Win32
10	Delphi 2006	<TurboDB 6 Installation>\Lib\Delphi10	Win32
11	Delphi 2007	<TurboDB 6 Installation>\Lib\Delphi11	Win32
12	RAD Studio / Delphi / C++ Builder 2009	<TurboDB 6 Installation>\Lib\Delphi12	Win32
14	RAD Studio / Delphi / C++ Builder 2010	<TurboDB 6 Installation>\Lib\Delphi14	Win32
15	RAD Studio / Delphi / C++ Builder XE	<TurboDB 6 Installation>\Lib\Delphi15	Win32
16	RAD Studio / Delphi / C++ Builder XE 2	<TurboDB 6 Installation>\Lib\Delphi16\win32 <TurboDB 6 Installation>\Lib\Delphi16\win64	Win32 Win64
17	RAD Studio / Delphi / C++ Builder XE 3	<TurboDB 6 Installation>\Lib\Delphi17\win	Win32 Win64

		32 <TurboDB 6 Installation>\Lib\Delphi17\win64	
18	RAD Studio / Delphi / C++ Builder XE 4	<TurboDB 6 Installation>\Lib\Delphi18\win32 <TurboDB 6 Installation>\Lib\Delphi18\win64	Win32 Win64
19	RAD Studio / Delphi / C++ Builder XE 5	<TurboDB 6 Installation>\Lib\Delphi19\win32 <TurboDB 6 Installation>\Lib\Delphi19\win64	Win32 Win64
20	RAD Studio / Delphi / C++ Builder XE 6	<TurboDB 6 Installation>\Lib\Delphi20\win32 <TurboDB 6 Installation>\Lib\Delphi20\win64	Win32 Win64
21	RAD Studio / Delphi / C++ Builder XE 7	<TurboDB 6 Installation>\Lib\Delphi21\win32 <TurboDB 6 Installation>\Lib\Delphi21\win64	Win32 Win64
22	RAD Studio / Delphi / C++ Builder XE 8	<TurboDB 6 Installation>\Lib\Delphi22\win32 <TurboDB 6 Installation>\Lib\Delphi22\win64	Win32 Win64
23	RAD Studio / Delphi / C++ Builder 10 Seattle	<TurboDB 6 Installation>\Lib\Delphi23\win32 <TurboDB 6 Installation>\Lib\Delphi23\win64	Win32 Win64
24	RAD Studio / Delphi / C++ Builder 10.1 Berlin	<TurboDB 6 Installation>\Lib\Delphi24\win32 <TurboDB 6 Installation>\Lib\Delphi24\win64	Win32 Win64
25	RAD Studio / Delphi / C++ Builder 10.2 Tokyo	<TurboDB 6 Installation>\Lib\Delphi25\win32 <TurboDB 6 Installation>\Lib\Delphi25\win64	Win32 Win64
26	RAD Studio / Delphi / C++ Builder 10.3 Rio	<TurboDB 6 InstallDir>\Lib\Delphi26\win32	Win32 Win64

<TurboDB 6
InstallDir>\Lib\Delphi26\win6
4

1.1.6.2 Lizenzierung und Aktivierung

Falls Sie die VCL/CLX Komponenten Bibliothek verwenden, müssen Sie die Unit *TdbLicense* in jedes Ihrer Projekte aufnehmen, um die lizenzierten Features zu aktivieren. Ohne diese Unit wird Ihr Programm eine Exception werfen, wenn eine Verbindung zur Datenbank aufgebaut werden soll. Das Einbinden von TdbLicense ist sehr einfach:

- In Delphi fügen Sie die Unit *TdbLicense* zur uses-Anweisung einer beliebigen Unit in Ihrem Programm hinzu.
- In C++ Builder fügen Sie die Zeile *#pragma link TdbLicense* in das Haupt C++ File Ihrer Anwendung ein.
  Prüfen sie zusätzlich die Projektoption "Laufzeit-Packages verwenden" (Standard bei C++ Builder). Deaktivieren Sie entweder diese Option oder fügen Sie die Lib-Datei "turbodb6dX.lib" (turbodb6dX.a bei x64) dem Projekt hinzu.

Solange Sie über keine Lizenzdatei verfügen und Ihre TurboDB-Version nicht aktiviert haben, werden Ihre Anwendungen 30 Tage, von Beginn der TurboDB-Installation an, lauffähig sein. Nach dieser Zeitspanne wird Ihre Anwendung eine Exception werfen und nicht mehr lauffähig sein.

Bei der Aktivierung wird aus einer Testversion eine lizenzierte TurboDB Version, die Sie zur Erstellung kommerzieller Anwendungen nutzen können. Um Ihre TurboDB Testversion zu aktivieren, benötigen Sie eine Lizenzdatei, die Sie beim Kauf der Lizenz erhalten.

Nehmen wir an Sie haben die Entwicklung Ihrer Anwendung mit der TurboDB Testversion abgeschlossen und kaufen nun eine Lizenz. Sie erhalten eine Email mit der Lizenzdatei.

1. Starten Sie das Programm Activator, das sich im bin Unterverzeichnis Ihrer TurboDB Installation befindet. Unter **Windows Vista oder höher** muss der Aktivator im Adminstrator-Modus ausgeführt werden. Starten Sie dazu den Aktivator nicht wie gewohnt durch Doppelklick, sondern indem Sie mit der rechten Maustaste das Kontextmenü öffnen und den Befehl *Als Administrator ausführen* wählen.
2. Prüfen Sie ob das angezeigte TurboDB Verzeichnis korrekt ist, geben Sie den Pfad zur Lizenzdatei ein und drücken Sie auf OK. Der Aktivator wird einige Dateien Ihrer TurboDB Installation modifizieren.
3. Fertig. Nochmal, vergessen Sie bitte nicht TdbLicense in allen Ihre TurboDB Anwendungen einzubinden. Das Modul ist sehr klein und beinhaltet eine verschlüsselte Lizenzinformation, die nötig ist um kommerziell benutzbare Programme zu erstellen.

1.1.7 Beispielprogramme

TurboDB für VCL wird mit einigen Demos ausgeliefert. Diese Projekte werden bei der Installation in den Ordner *Benutzer/Öffentlich* beziehungsweise i*dataweb/TurboDB VCL 6/Demos* kopiert.

Company: Master-Detail Datasets

ToDoList: SQL-Abfragen, Aufzählungen

Fulltext: Volltextindizierung und -suche

Relationship: Link und Relations Felder

Drill-Down: Selection API

Images: Blobs und die *TTdbBlobProvider* Komponente

1.1.7.1 Company Beispiel

Plattform: Windows

Sprache: Object Pascal

Compiler Version: 6 und höher

Beschreibung: Verwaltung von Angestellten und Ihren Abteilungen in zwei verknüpften Tabellensichten. Es kann gefiltert, gesucht, sortiert und exportiert werden.

Company ist eine VCL Anwendung und kann daher mit jedem Compiler übersetzt werden, der TurboDB unterstützt.

Demonstriert: Master-Detail Verknüpfung, Link Felder, Benutzung der TTdbBatchMove-Komponente

1.1.7.2 ToDoList Beispiel

Plattform: Windows

Sprache: C++

Compiler Version: 6

Beschreibung: Kleine & einfache Aufgabenverwaltung mit kurzer und ausführlicher Beschreibung, Deadline, Wichtigkeit uns Status. *ToDoList* ist ein CLX-Anwendung, Sie benötigen C++Builder 6 um es zu übersetzen.

Demonstriert: Queries, Datentype Aufzählung in Datenbanktabellen, Memos, Editieren sortierter Ergebnismengen, Benutzung von Look-up Feldern, Benutzung von *TTdbEnumValueSet*

1.1.7.3 Fulltext Beispiel

Plattform: Windows

Sprache: Object Pascal

Compiler Version: 6 und höher

Beschreibung: Erlaubt die Indizierung von Text Dateien aus auszuwählenden Verzeichnissen. Dann kann nach Schlüsselwörtern und Kombinationen aus Schlüsselwörtern gesucht werden. *Fulltext* ist eine VCL Anwendung und kann daher mit jedem Compiler übersetzt werden, der TurboDB unterstützt.

Demonstriert: Volltext-Indizierung und Volltext-Suche

1.1.7.4 Relationship Beispiel

Plattform: Windows

Sprache: Object Pascal

Compiler Version: 6 und höher

Beschreibung: Verwaltung von Abteilungen, Niederlassungen und Angestellten. Angestellte können Niederlassungen und diese wiederum Abteilungen zugeordnet werden. *RelationShip* ist eine VCL Anwendung und kann daher mit jedem Compiler übersetzt werden, der TurboDB unterstützt.

Demonstriert: Verwendung von Link- und Relations-Feldern zur Herstellung und Darstellung von 1:n und n:m Beziehungen. Verknüpfte Master-Detail Views.

1.1.7.5 Drill Down

Platform: Windows

Sprache: Object Pascal

Compiler Version: 6 und höher

Beschreibung: Darstellung einer Tabelle in einem Grid mit iterativer Selektion und Deselektion von Datensätzen. Ausgewählte Datensätze sind in der Indikator-Spalte gekennzeichnet. Der Anwender kann die Auswahl entweder durch Filterbedingungen oder manuell durchführen. Zu beachten ist, dass wegen der limitierten Fähigkeiten der VCL DBGrid Komponente, die Selektion nicht immer korrekt angezeigt wird. Drücken Sie den Refresh-Schalter zur Aktualisierung der Anzeige, falls nicht das erwartete Ergebnis zu sehen ist.

Demonstriert: Arbeiten mit Selektionen entweder durch Filterbedingungen oder durch manuelles Hinzufügen und Entfernen.

1.1.7.6 Images

Platform: Windows

Sprache: Object Pascal

Compiler Version: 6 und höher

Beschreibung: Speichert Bilder in eine Datenbanktabelle, zeigt sie an und erlaubt dem Anwender das Hinzufügen, Löschen und Ändern.

Demonstriert: Arbeit mit Blobs in der Datenbank und mit der Blobprovider Komponente.

Siehe auch

TTdbBlobProvider

1.1.8 Support

1.1.8.1 Support

dataweb bietet technische Unterstützung für seine Produkte über das Internet. Falls Sie Fragen betreffend TurboDB haben, versuchen Sie es mit einer der folgenden Angebote.

FAQ

Findet sich Ihr Problem in den FAQs, den häufig gestellten Fragen unter http://www.turbodb.de/de/support/faq_allgemein.html

Website

Besuchen Sie die TurboDB Website unter http://www.turbodb.de.

Forum

dataweb unterhält ein deutschsprachiges Diskussionsforum für TurboDB.

E-Mail

Falls Sie weitere Fragen, Bemerkungen oder Probleme haben, schreiben Sie uns eine E-Mail. Wir stehen Ihnen jederzeit gerne zur Verfügung: support@dataweb.de.

Dienstleistungen

Wir bieten Kompetenz in Beratung & Programmierung in den Bereichen TurboDB und allen anderen Bereichen der Windows-Programmierung mit C++, Delphi und C#. Wir sind Spezialisten für die Implementierung der Daten-Speicherung und -Abfrage, dem Design und der Implementierung von Skript-Sprachen, der Auswertung großer Mengen zeitabhängiger Daten (Zeitreihen), der Automation und Systemanalyse mit dem Einsatz editierbarer Diagramme: http://www.dataweb.de/de/produkte/diagramme.html

1.1.8.2 Versionen und Editionen

TurboDB 6 ist für Delphi/C++ Builder für Windows erhältlich.

Standard Edition

Die Standard Edition erlaubt es bis zu 63 Tabellen gleichzeitig zu öffnen und von verschieden Anwendungen aus zu bearbeiten.

Professional Edition

Zusätzlich zur Standard Edition verfügt die Professional über eine *TTdbQuery* Komponente, um mit SQL Anfragen auf Ihre Datenbank zuzugreifen.

Testversion

Die VCL Testversion bietet die Leistung der Professional Edition, allerdings nur 30 Tage ab Installation. Die Design-Time Unterstützung und mit dieser Edition erzeugte Anwendungen werden nach Ablauf dieser Frist die Arbeit einstellen.

1.2 VCL Komponenten Bibliothek

1.2.1 Überblick

Dieses Kapitel beschreibt die Komponenten Bibliothek für Delphi und C++ Builder. Für detaillierte Information zur Datenbank Engine, seinen Features, oder dem Turbo SQL Dialekt, lesen Sie bitte das Kapitel TurboDB Engine.

1.2.2 Entwickeln mit TurboDB

1.2.2.1 Mit Tabellen arbeiten

1.2.2.1.1 Eine Tabelle zur Entwurfszeit erzeugen

Es gibt mehrere Möglichkeiten eine neue Datenbanktabelle zu erzeugen. Sie können die Werkzeuge benutzen, die direkt in die Delphi IDE eingebaut sind, Sie können den TurboDB Viewer verwenden oder Sie greifen auf das textbasierte Werkzeug TurboDB Workbench zurück.

Um eine Datenbanktabelle in der IDE zu erzeugen müssen Sie:

1. Eine *TTdbTable* Komponente auf ein Formular oder Datenmodul legen
2. Mit der rechten Maustaste auf die Komponente klicken und den Menüpunkt *New Table* auswählen
3. Die Tabellenspalten definieren und die Tabelle erzeugen.

Um eine Datenbanktabelle mit TurboDB Viewer zu erzeugen müssen Sie:

1. Öffnen Sie TurboDB Viewer über das Delphi Tools Menü oder das Start Menü.
2. Wählen Sie das Kommando *Database/New/Table*...
3. Editieren Sie die Tabellenspalten und erzeugen Sie die Tabelle.

1.2.2.1.2 Eine Tabelle zur Laufzeit erzeugen

Die CreateTable Methode der TTdbTable Komponente erstellt eine neue Datenbanktabelle zur Laufzeit. Die Tabellenstruktur wird über die FieldDefsTdb Eigenschaft der Tabelle festgelegt. Falls Sie eine komplett neue Tabelle erzeugen möchten, müssen Sie FieldDefsTdb erst leeren bevor Sie neue Felder hinzufügen.

TurboDB unterstützt auch den Standard Mechanismus basierend auf dem FieldDefs Eigenschaft von TDataSet. Während dies ein guter Weg ist um Kompatibilität zu gewährleisten, bietet FieldDefsTdb eine bessere Kontrolle über die spezielle Möglichkeiten und Feldtypen von TurboDB, die nicht von TDataSet Standard unterstützt werden.

Um FieldDefsTdb zu leeren,

- Verwenden Sie *FieldDefsTdb.Clear*.

Um ein TdbFieldDef zu den FieldDefsTdb einer Tabelle hinzuzufügen,

1. Verwenden Sie *FieldDefsTdb.Add*,
2. Setzen Sie anschließend die Eigenschaften von *TdbFieldDef*, das von dieser Funktion zurückgegeben wird.

Um eine Datenbanktabelle zur Laufzeit zu erzeugen,

1. Bestimmen Sie die *FieldDefsTdb* Eigenschaft entsprechend der Spalten, über die die neue Tabelle verfügen soll,
2. Legen Sie den *TableLevel* der Tabelle fest,
3. Bestimmen Sie Verschlüsselungsmethode und *Password*, falls Sie Ihre Tabelle schützen wollen,
4. Setzen Sie die Eigenschaften *DatabaseName* und *TableName*
5. Rufen Sie die Methode *CreateTable* auf.

Anmerkung: Falls schon eine Tabelle mit diesem Namen existiert, wird diese nicht überschrieben sondern es erscheint eine Exception, die über den Misserfolg der Aktion informiert

Der folgende Programmausschnitt erzeugt einer Tabelle mit drei Spalten:

```
TdbTable2.Close;
TdbTable2.FieldDefsTdb.Clear;
TdbTable2.FieldDefsTdb.Add('Word', dtString);
TdbTable2.FieldDefsTdb.Add('Count', dtSmallInt);
with TdbTable2.FieldDefsTdb.Add('RecordId', dtAutoInc) do Specification
:= 'Word';
TdbTable2.TableName := 'index';
TdbTable2.TableLevel := 4;
TdbTable2.CreateTable;
TdbTable2.Open;
```

Um eine berechnete Spalte anzulegen

1. Bei Definition der Tabellenspalten mit *TTdbFieldDef* Objekten, weisen Sie der Eigenschaft *CalcExpression* eine Berechnungsvorschrift zu.
2. Falls die Berechnung immer durchgeführt werden soll wenn sich die Daten eines Datensatzes ändern, ist die Eigenschaft *InternalCalcField* auf *True* zu setzen. Falls die Berechnung nur für den Vorgabewert der Spalte verwendet werden soll, ist *InternalCalcField* auf *False* zu setzen.

1.2.2.1.3 Eine Tabelle zur Laufzeit ändern

Verwenden Sie die AlterTable Methode um eine Datenbanktabelle zur Laufzeit zu restrukturieren. AlterTable verwendet die FieldDefsTdb Eigenschaft um die Struktur der geänderten Tabelle festzulegen. TurboDB übernimmt wenn möglich den Datenbestand, sogar dann, wenn sich der Feldtyp ändert oder wenn das Feld umbenannt wird. Die Tabellenstruktur ist genauso festzulegen wie beim Erzeugen einer Tabelle. AlterTable kann auch zum Ändern des Tabellenlevels oder des Passworts bzw. der Verschlüsselungsmethode verwendet werden.

Bemerkung: Das Ändern der Tabellenstruktur kann zu Datenverlusten führen, speziell wenn Sie Tabellenspalten entfernen oder die Länge alphanummerischer Felder verkürzen.

Bemerkung: Falls während der Restrukturierung ein Problem auftritt oder Ihr Programm abstürzt, sind Ihre Daten nicht verloren. Die Originaltabellen werden in ~TabellenName umbenannt, bevor die Restrukturierung beginnt. Falls eine Problem auftritt, können Sie diese Dateien einfach umbenennen, um den alten Stand wiederherzustellen.

Das Beispiel zeigt, wie die Verschlüsselung einer Tabelle zur Laufzeit geändert werden kann:

```
TdbTable4.EncryptionMethod := temBlowfish;
TdbTable4.Password := 'dataweb';
TdbTable4.AlterTable;
```

1.2.2.1.4 Selektionen und Drill-Down

TurboDB verfügt über eine Erweiterung der konventionellen Datenbank-Filter der VCL. Eine Selektion ist eine Untermenge aller Datensätze, die durch Hinzufügen oder Entfernen einzelner Datensätze manipuliert werden kann, indem Filter-Bedingungen und Wort-Filter angewendet werden. Die Untermenge kann beispielsweise zur Verwaltung der Mehrfachselektion in einem Daten-Grid verwendet werden, oder um Drill-Down Fähigkeiten oder die Funktionalität der SQL Operatoren UNION, INTERSECT und EXCEPT auf *TTdbTable* Level zu realisieren. Selektionen können natürlich auch als Filter verwendet werden.

Filter über die Datensätze mit den Nummern 54821 und 897003:

```
MyTable.AddToSelection(54821);
MyTable.AddToSelection(897003);
MyTable.Filtered := True;
```

Filter über alle Autos der Marke BMW und inkrementell mit roter Farbe (drill-down):

```
MyTable.Filter = 'Manufacturer = ''BMW''';
MyTable.Filtered := True;
// Now all BMWs are shown
MyTable.AddToSelection('Color = ''red''');
// Now all red BMWs are in the selection
MyTable.ActivateSelection;
// Now all red BMWs are shown
MyTable.Filtered := False;
// All cars are shown again.
```

Löschen aller roter BMWs (inkrementell):

```
MyTable.ClearSelection;
MyTable.AddToSelection('Manufacturer = ''BMW''');
MyTable.IntersectSelection('Color = ''red''');
MyTable.Last;
while not MyTable.BOF do begin
  if MyTable.IsSelected(MyTable.RecNo) then
    MyTable.Delete;
  MyTable.Prior;
end;
```

Falls ein Filter für eine Datenmenge definiert ist, die Eigenschaft Filtered aber False ist, besteht die aktuelle Selektion aus den Datensätzen, die die Filterbedingung erfüllen. Mit den Methoden *AddToSelection*, *RemoveFromSelection* und *IntersectSelection* kann diese bearbeitet werden. Die Methoden *FindFirst* und *FindNext* können zum Durchwandern der aktuellen Selektion verwendet werden.

See also

DrillDown Demo Program
AddToSelection method
RemoveFromSelection method
IntersectSelection method
ActivateSelection method
IsSelected method
ClearSelection method

1.2.2.2 Indexe benutzen

1.2.2.2.1 Einen Index zur Entwurfszeit erzeugen

Es gibt verschiedene Möglichkeiten einen Index zu erstellen. Sie können die in die Delphi IDE integrierten Werkzeuge benutzen, Sie können den TurboDB Viewer verwenden oder Sie nutzen die textbasierte TurboDB Workbench.

Um einen Index in der IDE zu erstellen:

1. Ziehen Sie eine *TTdbTable* Komponente auf ein Formular oder ein Datenmodul
2. Klicken Sie mit der rechten Maustaste auf die Komponente und wählen Sie den Menüpunkt *Properties*
3. Definieren Sie den Index und erzeugen Sie ihn.

Um einen Index mit TurboDB Viewer zu erstellen:

1. Starten Sie TurboDB Viewer über das Delphi Tools Menü oder das Start Menü.
2. Öffnen Sie eine Tabelle über *Database/Open/Table*
3. Wählen Sie *Table/Properties...*
4. Definieren Sie den Index und erzeugen Sie ihn.

1.2.2.2.2 Einen Volltextindex zur Entwurfszeit erzeugen

Ein Volltextindex ist eine Datenstruktur, die es Ihnen ermöglicht über Schlüsselwörter, die sich in jeder Spalte der Tabelle befinden können, einen Datensatz sehr schnell zu finden. TurboDB realisiert dieses Feature indem eine Hilfstabelle erzeugt wird, die Wörterbuch genannt wird und alle Schlüsselwörter enthält. In älteren Tabellen (Level 3 und niedriger) wurde die Beziehung zwischen dieser Schlüsselworttabelle und der Originaltabelle mit einem Relations-Feld hergestellt. In aktuellen Tabellen-Leveln gibt es einen speziellen Volltext-Index, der diese Beziehung auf einem wesentlich schnelleren und wartbaren Weg herstellt.

Um einen Volltextindex mit TurboDB Viewer zu erstellen (Tabellen-Level 4 und höher):

1. Starten Sie TurboDB Viewer und öffnen Sie die Datenbank in der Sie einen Volltext-Index erstellen möchten.
2. Erstellen Sie die Wörterbuchtabelle falls noch keine existiert. Sie Tabelle muss eine String-Spalte haben, die lang genug ist um die Schlüsselwörter aufnehmen zu können (z.B. 40 Zeichen), eine Spalte von Typ Byte, die später die Relevanz eines Schlüsselwortes speichern soll, und eine AutoInc-Spalte mit der Schlüsselwort-Spalte als Anzeigeinformation (Indication).
3. Wählen Sie *Table\Properties...* um das Eigenschaften-Fenster für die Tabelle anzuzeigen, auf die der Volltext-Index anzuwenden sein soll. Beachten Sie, dass der neuen Volltext-Index-Typ nur für Tabellen-Level 4 und höher verfügbar ist.
4. Wechseln Sie auf die Index-Seite und klicken Sie auf den *New*-Knopf.
5. Vergeben Sie einen Namen für den Index und legen Sie Volltext-Index als Indextyp fest.
6. auf der Volltext-Seite wählen Sie alle Spalten aus, die in den Volltext-Index eingehen sollen und legen Sie die zu verwendende Wörterbuchtabelle fest (die in Schritt 2 erstellt wurde).
7. Belassen Sie die minimale Relevanz auf dem vorgegebenen Wert und bestätigen Sie mit Ok. Der Volltext-Index wird erstellt.

Um einen Volltextindex mit TurboDB Viewer zu erstellen (Tabellen-Level 3 und niedriger):

1. Starten Sie TurboDB Viewer. Normalerweise finden Sie einen entsprechenden Eintrag im Tools Menü. Die Anwendung befindet sich im bin Verzeichnis Ihrer TurboDB Installation.
2. Öffnen Sie die Tabelle für die ein Volltextindex erstellt werden soll mit dem Kommando *Database/Open/Table...*
3. Starten Sie den Volltextindex Assistenten mit *Table/Create Full-Text Index...*

4. Auf der ersten Seite legen Sie einen Namen für die Hilfstabelle zur Aufnahme der Schlüsselwörter fest. Geben Sie dann einen Namen für die Schlüsselwort Spalte in der Originaltabelle ein. Als Drittes können Sie noch die maximale Länge der Schlüsselwörter bestimmen. Schlüsselwörter die diesen Wert übersteigen werden abgeschnitten.
5. Auf der zweiten Seite legen Sie die Spalten der Originaltabelle fest, die bei der Bildung des Volltextindex berücksichtigt werden sollen. Sie können alle Spalten verwenden, sehr oft sollen aber nur bestimmte Spalten eingefügt werden.
6. Da die Suche nach Wörtern wie und, in oder ein in einem Volltextindex wenig Sinn macht, sollten Sie versuchen Wörter dieser Art vom Volltextindex auszuschließen. Dies kann auf zwei Arten geschehen. Zum einen können Sie Schlüsselwörter nicht berücksichtigen, die öfter als ein festzulegendes Limit (z.b. 100) auftreten. Zum zweiten können Sie eine Textdatei mit den Wörtern definieren, die nicht in den Volltextindex aufgenommen werden sollen. Beide Techniken helfen den Volltextindex schlank zu halten, erhöhen die Suchgeschwindigkeit und vermeiden unnütze Treffer. Die entsprechenden Einstellungen sind auf der dritten Seite des Assistenten zu machen.
7. Auf der vierten und letzten Seite sind Sie schließlich bei der Erstellung des Volltextindex angelangt. Da die Aktion das Scannen aller Felder und aller Datensätze der Tabelle umfasst, kann bei einem großem Datenbestand bis zur Fertigstellung schon eine Weile vergehen.

1.2.2.2.3 Einen Volltextindex zur Laufzeit erzeugen

Es können nur Volltext-Index vom neuen Typ, also für Tabellen-Level 4 und höher, zur Laufzeit erstellt werden.

Um eine neue Tabelle mit Volltext-Index mit einer TTdbTable Komponente zu erstellen:

1. Legen Sie eine neue *TTdbTable* Komponente für die Tabelle an, die Sie erstellen möchten.
2. Definieren Sie alle für die Tabelle benötigten Eigenschaften (z.B. FieldDefs).
3. Fügen Sie ein *TTdbFulltextIndexDef* für jeden Volltext-Index zur *FulltextIndexDefs* Property der Tabelle hinzu.
4. Verwenden Sie die *CreateTable* Methode der *TTdbTable* Komponente.

Einen Volltext-Index zur Laufzeit mit einer *TTdbTable* Komponente erstellen:

1. Erzeugen Sie eine *TTdbTable* Komponente für die Tabelle, für die ein neuer Volltext-Index erstellt werden soll.
2. Verwenden Sie die *AddFulltextIndex* Methode, mit den oben beschriebenen Parametern.

Einen Volltext-Index zur Laufzeit mit SQL erstellen:

1. Erzeugen Sie eine *TTdbQuery* Komponente und geben Sie diese SQL Befehl an:
```
CREATE FULLTEXTINDEX ON <Tabellen-Name> DICTIONARY <Wörterbuch-
Tabellen-Name> (<Feld1>, <Feld2>, <Feld3>, ...)
```
2. Verwenden Sie die Methode *ExecSQL* der *TTdbQuery* Komponente.

Siehe auch

CREATE FULLTEXTINDEX

1.2.2.2.4 Einen Volltextindex zur Laufzeit benutzen

Nach dem Erstellen einen Volltextindex können Sie Datensätze, die eines oder mehrere Schlüsselwörter enthalten, in Sekundenbruchteilen finden.

Um einen Volltextindex zur Laufzeit zu benutzen, müssen Sie:

1. Platzieren Sie eine *TTdb*Table Komponente für die Tabelle in der Sie suchen wollen auf Ihrem Formular.
2. Setzen Sie zur Entwurfs- oder Laufzeit in die Eigenschaft *WordFilter* auf den Suchausdruck, z.B: `WordFilter := 'Embarcadero';`

Die Syntax für Volltext-Filterausdrücke ist in "Volltext Suchbedingungen"beschrieben.

3. Nun können Sie mit dem Filter arbeiten wie Sie es von herkömmlichen Filter gewohnt sind. Sie können die *FindFirst*, *FindNext*, *FindPrior* und *FindLast* Methoden anwenden um den gesuchten Datensatz zu finden oder Sie setzen einfach die *Filter* Eigenschaft auf True.
4. Es ist auch möglich Volltext-Filter mit herkömmlichen Filter zu kombinieren, indem gleichzeitig *WordFilter* und *Filter* (oder *FilterW*) mit Werten belegt werden.

Anmerkung:

Für einen Volltext-Index der "alten Art" (TabellenLevel 3 und kleiner) benötigen Sie auch eine Tabellen-Komponente für die Wörterbuchtabelle mit den Schlüsselwörtern. Setzen Sie die *FullTextTable* Eigenschaft der Tabellenkomponente auf die Komponente der Wörterbuchtabelle.

1.2.2.2.5 Einen Index aktualisieren oder reparieren

Manchmal, hauptsächlich auf Grund von Programmabstürzen oder wegen Abbrüchen während dem Debuggen, kann ein Index beschädigt werden und dann nicht mehr synchron mit der Tabelle sein. Sie bemerken dies, wenn die Anzahl der Datensätze nach dem Setzen des Index nicht mit der erwarteten übereinstimmt oder bei einer Überprüfung der Tabelle mit TurboDBViewer. Ist ein Index nicht mehr korrekt, kann er einfach durch Neuaufbau repariert werden:

Neuaufbau des Index mit TurboDB Viewer:

1. Öffnen Sie die Datenbank und wählen Sie die Tabelle aus für die ein Index zu reparieren ist.
2. Wählen Sie *Table\Maintain...* aus dem Hauptmenü.
3. Falls Sie zuerst die Tabelle prüfen möchten, klicken Sie auf den *Start* Knopf und warten Sie bis die Analyse abgeschlossen ist. Falls für die Tabelle ein Problem existiert, wird es angezeigt. Sollte ein Index betroffen sein, muss dieser neu aufgebaut werden.
4. Aktivieren Sie die Option *Rebuild all indexes* in der *Repair Options* Gruppe und klicken Sie auf *Apply*. TurboDB Viewer wird nun alle Index der Tabelle neu aufbauen. Das kann bei großen Tabellen einige Minuten in Anspruch nehmen.

Neuaufbau eines Index zur Laufzeit mit einer TTdbTable Komponente:

1. Erzeugen Sie eine *TTdbTable* Komponente für die Tabelle für die ein Index wiederhergestellt werden soll.
2. Falls es sich bei dem Index um einen normalen Index handelt, verwenden Sie die Methode *UpdateIndex*, falls es ein Volltext-Index ist, rufen Sie *UpdateFulltextIndex* auf.

Neuaufbau eines Indes zur Laufzeit mit SQL:

1. Erstellen Sie eine *TTdbQuery* Komponente für die Datenbank in der sich der Index befindet.
2. Setzen Sie die Eigenschaft SQL.Text auf
```
UPDATE INDEX <Tabellen-Name>.<Index-Name>
bzw. bei einem Volltextindex auf
UPDATE FULLTEXTINDEX <Tabellen-Name>.<Volltext-Index-Name>
```
3. Rufen Sie die ExecSQL Methode der *TTdbQuery* Komponente auf.

1.2.2.3 Datensätze importieren und exportieren

1.2.2.3.1 Einen Batch-Move ausführen

Die Batch-Move Komponente stellt den leistungsstarken und flexiblen Weg dar, um Datensätze von und zu unterschiedlichen Datenquellen zu transferieren. Sie können verschiedene Dateitypen und alle TDataSet Nachkommen als Datenquelle verwenden.

Um einen Batch-Move auszuführen:

1. Setzen Sie die TdbDataSet Eigenschaft auf die TTdbTable Komponente in die Sie importieren oder von der Sie exportieren möchten.

2. Setzen Sie entweder FileName oder DataSet auf die Datei oder das Dataset der anderen Datenquelle.
3. Falls die andere Datenquelle eine Datei ist, setzen Sie die FileType Eigenschaft in Übereinstimmung zu dem Dateiformat der anderen Datenquelle.
4. Falls die andere Datenquelle eine Textdatei ist (z.B. FileType = tffSDF), setzen Sie Quote, Separatorund ColumnNames auf die entsprechenden Werte.
5. Falls die andere Datenquelle eine Textdatei oder eine dBase Datei ist, setzen Sie CharSet auf den entsprechenden Wert.
6. Setzen Sie die Direction Eigenschaft auf bmdImport, falls Daten von der anderen Datenquelle in die TurboDB Tabelle übertragen werden sollen. Setzen Sie es auf bmdExport, falls Sie eine Datei mit den Daten der TurboDB Tabelle erzeugen möchten.
7. Setzen Sie die Mode Eigenschaft auf den Typ des Batch-Moves den Sie durchführen möchten.
8. Fügen Sie Spaltenzuweisungen zwischen Quelle und Ziel zu der Mappings Eigenschaft hinzu.
9. Verwenden Sie die Filter Eigenschaft um den Import bzw. Export auf eine bestimmte Untermenge der Quelldaten zu beschränken.
10. Registrieren Sie eine Ereignisbehandlungsroutine für das OnProgress Ereignis um über den Fortschritt des Batch-Move Prozesses informiert zu sein und dem Anwender einen Weg anzubieten die Aktion abzubrechen.
11. Rufen Sie Execute auf um den Batch-Move zu starten.

1.2.2.4 Portierung von BDE

1.2.2.4.1 Portierung einer BDE Anwendung nach TurboDB

Die Portierung von BDE Anwendungen, die mit Paradox oder dBase Tabellen arbeiten, nach TurboDB ist sehr einfach, da TurboDB auf Komponenten basiert, die sehr ähnlich zu den vergleichbaren BDE Komponenten sind.

TTdbDataSet ersetzt *TBDEDataSet*, *TTdbTable* ersetzt *TTable*, *TTdbQuery* ersetzt *TQuery*, *TTdbDatabase* ersetzt *TDatabase* und *TTdbBatchMove* ersetzt *TBatchMove*. Diese TurboDB Komponenten bieten die selben Eigenschaften, Methoden und Ereignisse wie die BDE Komponenten, so dass sie einen Großteil Ihres existierenden Quelltextes mit TurboDB wiederverwenden können.

Im Wesentlichen funktioniert die Portierung einer BDE Anwendung so:

a) Nutzung des Konvertierungsassistenten der Tabelle (nur verfügbar in Delphi/C++ Versionen mit integrierter BDE)

1. Installieren Sie TurboDB in die Delphi IDE.
2. Erstellen Sie eine Sicherung Ihres BDE Projektes und laden Sie es anschließend in die Delphi IDE.
3. Platzieren Sie für jede *TTable* eine *TTdbTable* Komponente in Ihr Formular oder Datenmodul.
4. Klicken Sie mit der rechten Maustaste auf jede TTdbTable Komponente und wählen Sie das Convert BDE Table.. Kommando.
5. Selektieren Sie die *TTable* Komponente, die Sie konvertieren möchten.
6. TurboDB wird eine passende Tabelle anlegen und die Daten der BDE Tabelle übernehmen. Die Eigenschaften der BDE Komponente werden ebenso übernommen.
7. Bennenen Sie alle *TTable* Komponenten um und geben Sie den *TTdbTable* Komponenten die Originalnamen der ursprünglichen *TTable* Komponenten.
8. Falls Ihre Anwendung *TQuery* Objekte enthält, ersetzen Sie diese mit *TTdbQuery*. Unter "Turbo SQL vs. Local SQL"finden Sie kleinere zu beachtende Unterschiede in der SQL Syntax.
9. Falls Ihr Projekt ein *TDatabase* und/oder *TSession* Objekt beinhaltet, ersetzen Sie es mit einem

TTdbDatabase Objekt.

10. Falls Ihr Projekt mit einer *TBatchMove* Komponente arbeitet, benutzen Sie eine TTdbBatchMove Komponente stattdessen.
11. Nun kompilieren und starten Sie Ihre Anwendung. Falls Sie Fehlermeldungen vom Compiler oder Ihrer Anwendung erhalten, werfen Sie bitte einen Blick in das Kapitel "Unterschiede zwischen BDE und TurboDB".

b) Manuelle Konvertierung

1. Öffnen Sie den TurboDB Viewer und konvertieren Sie im ersten Schritt alle Tabellen, die Sie im Projekt benötigen nach TurboDB (Menüpunkt Tools/Import Tables). Kontrollieren Sie dabei im Viewer, ob die Inhalte auch richtig übernommen wurden.
2. Installieren Sie TurboDB in die Delphi IDE.
3. Erstellen Sie eine Sicherung Ihres BDE Projektes und laden Sie es anschließend in die Delphi IDE.
4. Platzieren Sie für jede *TTable* eine *TTdbTable* Komponente in Ihr Formular oder Datenmodul und öffnen Sie in der TurboDB-Table Komponente die passende in Schritt 1. konvertierte Tabelle.
5. Legen Sie in der *TDBTable* nun noch folgende Properties fest: AutoCalcFields, Exclusive, Filter, FilterOptions, ReadOnly, IndexName, Filtered.
   Orientieren Sie sich dabei an den gleichnamigen Einstellungen der BDE-Table Komponente.
6. Gehen Sie gleichermassen bei den Events der TurboDB-Table Komponente vor
7. Bennenen Sie alle *TTable* Komponenten um und geben Sie den *TTdbTable* Komponenten die Originalnamen der ursprünglichen *TTable* Komponenten.
8. Falls Ihre Anwendung *TQuery* Objekte enthält, ersetzen Sie diese mit *TTdbQuery*. Unter "Turbo SQL vs. Local SQL"finden Sie kleinere zu beachtende Unterschiede in der SQL Syntax.
9. Falls Ihr Projekt ein *TDatabase* und/oder *TSession* Objekt beinhaltet, ersetzen Sie es mit einem TTdbDatabase Objekt.
10. Falls Ihr Projekt mit einer *TBatchMove* Komponente arbeitet, benutzen Sie eine TTdbBatchMove Komponente stattdessen.
11. Nun kompilieren und starten Sie Ihre Anwendung. Falls Sie Fehlermeldungen vom Compiler oder Ihrer Anwendung erhalten, werfen Sie bitte einen Blick in das Kapitel "Unterschiede zwischen BDE und TurboDB".

1.2.2.4.2 Unterschiede zwischen BDE und TurboDB

- TurboDB verwendet ein eigenes Dateiformat. Das bedeutet, dass Sie Ihre Tabellen konvertieren müssen, wie es in "Portierung einer BDE Anwendung nach TurboDB" beschrieben ist.
- TurboDB unterstützt keine SQL Server. Falls Sie eine BDE Anwendung, die auf einen SQL Server zugreift, nach TurboDB konvertieren möchten, müssen Sie für Ihre Daten TurboDB Tabellen erzeugen wie es in "Portierung einer BDE Anwendung nach TurboDB" beschrieben wird.
- TurboDB unterstützt einige unter BDE verfügbare Feldtypen nicht. Dies sind: ftWord, ftCurrency, ftBCD, ftParadoxOle, ftDBaseOle, ftTypedBinary, ftCursor, ftFixedChar, ftWideString, ftLargeint, ftADT, ftArray, ftReference, ftDataSet, ftOraBlob, ftOraClob, ftVariant, ftInterface, ftIDispatch, ftGuid
- Der SQL Dialekt, der unter TurboDB (Turbo SQL) zum Einsatz kommt, kennt keine Outer Joins. Unter "Turbo SQL vs. Local SQL" finden Sie Details.
- Es gibt kein *TSession* Objekt in TurboDB. Wie dbExpress und andere Delphi Datenbank Technologien verwendet TurboDB Connection Objekte (*TTdbDatabase*) zur Verwaltung der

Verbindung.

- Der TurboDB Sprachtreiber arbeitet durch Angabe einer Windows Kollation.

1.2.2.4.3 Mehr als die BDE

TurboDB bietet Ihnen einige Möglichkeiten, die über das hinaus gehen was Sie mit der BDE machen können:

- Passwortschutz und Verschlüsselung Ihrer Tabellen
- Auffinden von Datensätzen durch Schlüsselwortsuche über jedes beliebige Feld in Sekundenbruchteilen mit und, oder und nicht Operatoren
- Wesentlich flexiblere Batch-Move Komponente zum Datentransfer von und zu jedem beliebigen Dataset und zu MyBase Datenpaketen
- Hinzufügen, Ändern und Löschen von Tabellenspalten zur Laufzeit mit der leistungsstarken AlterTable Methode
- Verwenden Sie Bindestriche und deutsche Umlaute in Tabellen und Spaltennamen
- Verwenden Sie Aufzählungstypen als Datentyp für Tabellenspalten
- Benutzerfreundliche, leicht zu bedienende visuelle Datenbankoberfläche mit einem leistungsfähigen Import und Export Assistenten
- Spezielle Spaltentypen zur einfachen, intuitiven Implementierung von eins-zu-viele und viele-zu-viele Beziehungen zwischen Tabellen

1.2.2.5 Lokalisierung Ihrer Anwendung

1.2.2.5.1 TurboDB Fehlermeldungen anpassen

TurboDB produziert drei Arten von Fehlermeldungen: Kontextmeldungen, Fehlerbeschreibungen und Meldungen die den Grund eines aufgetretenen Fehlers beschreiben. Alle Meldungstexte befinden sich in der Object Pascal Unit TdbMessages.pas. Diese Unit wird mit jeder Edition von TurboDB im Quelltext ausgeliefert. Im Auslieferungszustand sind alle Meldungstexte in Englisch und Deutsch vorhanden. Die Standardsprache ist Englisch.

Falls Sie deutsche Meldungen verwenden möchten müssen Sie:

1. Das Symbol GERMAN in den Bedingungen Ihrer Projektoptionen definieren.
2. Vergewissern Sie sich, dass TdbMessages.pas im Unit Suchpfad enthalten ist.
3. Falls Sie C++ Builder verwenden müssen Sie TdbMessages.pas zusätzlich in das Projekt aufnehmen.

Falls Sie die Meldungen in einer anderen Sprache haben möchten, können Sie sie selbst übersetzen und Sie in die Unit TdbMessages.pas einfügen, begleitet mit einer passenden bedingten Compiler-Direktive. Wenn Sie uns Ihre Übersetzung der Meldungen schicken, werden wir Sie in die Quellen einbauen. Auf diese Weise können auch andere von Ihrer Arbeit profitieren und Sie müssen den Quellcode nicht mit jeder neuen TurboDB Version anpassen.

1.2.2.5.2 Sprachspezifischer Zeichenkettevergleich

Länder, die einen anderen Zeichensatz verwenden als England oder Deutschland benötigen angepaßte Sortiersequenzen.

Sprachtreiber die bis TurboDB 5 eingesetzt wurden sind obsolet und sollen nicht mehr verwendet werden. TurboDB 6 unterstützt Windows Kollationen die auf Tabellen- und/oder Spaltenebene gesetzt werden können. Siehe dazu Kollationen und TTdbTable.Collation.

1.2.2.6 Verschiedenes

1.2.2.6.1 Unicode Zeichenketten speichern

Bis zu den 2007er Versionen von Delphi und C++Builder ist der Standard-String ein ANSI-String, der einfach in String- und Memo-Spalten gespeichert werden kann. In diesen Versionen müssen Unicode-Strings als WideString definiert werden und es ist nicht ganz so einfach, diese korrekt in die Datenbank zu schreiben. Beginnend mit Delphi und C++Builder 2009 ist der Standard-String ist Unicode und es führt zu Schwierigkeiten, wenn Sie sie als (ANSI) String oder Memo speichern.

Speichern von Unicode Strings mit Delphi/C++ Builder 2007 und früher

Die TurboDB-Engine erlaubt Unicode in WideString Spalten und in WideMemos. Bei der Verwendung dieser Datentypen in einer VCL Anwendung gilt es allerdings einige Fallen zu beachten. Die VCL verwendet in beinahe allen Funktionen und Komponenten AnsiStrings und konvertiert Unicode Zeichenketten automatisch und ohne Warnung in diesen Typ.

Datenbank-Kontrollelemente verarbeiten wiederum nur AnsiStrings. Es gibt keinen Weg Unicode mit VCL Daten-Kontrollelementen in Ihre Datenbank einzugeben oder anzuschauen.

Das können Sie mit Unicode Daten aus Ihrer TurboDB-Datenbank in Ihrer VCL-Anwendung machen:

- Anzeige und Eingabe von Zeichenketten in normale Kontrollelemente wie zum Beispiel *TEdit*. Diese Kontrollelemente arbeiten auch mit Unicode korrekt, Sie müssen aber Ihre eigene Logik für Updates und Postings implementieren, da normale Kontrollelemente über keinen *DataSource* Link verfügen:
- Interne Verwendung von Unicode Daten. Sie können *WideStrings* aus einem DataSet lesen und zurückschreiben.
- Falls Sie über *TdbDataSet* auf Unicode Daten zugreifen, verwenden Sie nicht die *TField.AsString* sondern die Eigenschaft *Value* oder das Property *AsWideString*.
- Query Parameter (z.B. Einträge in der *TTdbQuery.Params* Kollektion) können nur über *TParam.Value* gesetzt werden.
- Falls Sie mit Unicode Memos arbeiten möchten, werden Sie schnell merken, dass VCL keinen passenden Datentypen für diese Art von Tabellen-Spalte bietet. Unicode Memos werden durch den Feldtyp *ftBlob* und der Feldklasse *TTdbBlobField* repräsentiert, die eine Eigenschaft *AsWideString* bietet.
- Die Eigenschaft *Filter* in VCL Komponenten ist ebenso wie die SQL Eigenschaft vom Datentyp *AnsiString* und können daher auch nicht für die Suche in Unicode Zeichenketten verwendet werden. Die TurboDB Datenbank-Komponenten verfügen daher über die zusätzlichen Eigenschaften *FilterW* und *SQLW* vom Type *WideString*.

Speichern von ANSI Strings mit Delphi/C++ Builder 2009 und höher

Viele Anwendungen die nicht sprachunabhängig sein müssen, können Ihre Text-Daten weiterhin in String und Memo-Spalten speichern, mit zwei Vorteilen:

- Bessere Kompatibilität mit älteren Anwendungen
- Weniger Speicherverbrauch und damit einhergehend eine bessere Performanz.

Allerdings erfordert das Speichern von (UNICODE) Strings in (ANSI) String oder Memo-Spalten eine Konvertierung der Delphi Variablen nach *AnsiString*. Es muss sichergestellt sein, dass der Inhalt der *UnicodeString* Variable keine echten Unicode Zeichen enthält. Andernfalls kommt es beim Speichern des Strings in die Datenbank zu Informationsverlust.

1.2.2.6.2 Geschützte Datenbank-Tabellen

Beim Öffnen geschützter Tabellen müssen Sie ein Passwort angeben. Es gibt verschiedene Möglichkeiten dies zu tun:

a) Setzen Sie *EncryptionMethod* und *Password* von *TTdbTable* bevor Sie die Open-Methode aufrufen:

b) Definieren Sie eine Ereignisbehandlungsroutine für *TTdbDatabase.OnPassword*. Diese Routine wird aufgerufen, wenn TurboDB eine geschützte Tabelle nicht öffnen kann. Sie können das Passwort in dieser Routine bereitstellen.

c) Nehmen Sie die Unit *TdbPasswordDlg* in Ihr Projekt auf. Die Unit beinhaltet einen einfachen Passwort-Dialog, der in der Datenbankzugriffs-Palette registriert wird und jedes mal angezeigt wird, wenn ein Passwort benötigt wird. Dieser Passwort-Dialog wird auch zur *Entwurfszeit* verwendet.

Siehe auch

Data Security

1.2.2.6.3 Read-Only Tables and Databases

Eine Datenbank und/oder eine Tabelle kann im Read-Only Modus geöffnet werden, um jeglichen modifizierenden Zugroiiff darauf auszuschließen. Auch in diesem Modus muss TurboDB eine net-Datei erzeugen, andernfalls könnte eine zweite Applikation die Datenbank mit Schreiberlaubnis öffnen und einen Zugriffskonflikt auslösen.

Falls sich die Datenbank an einem Ort befindet an dem keine Schreibzugriff möglich ist, z.B. eine DVD oder ein geschütztes Verzeichnis, mus die Datenbank exklusiv und read-only geöffnet werden. In diesem Fall wird TurboDB nicht versuchen net-Dateien anzulegen.

In den seltenen Fällen, in denen mehrere Anwendungen gleichzeitigen Zugriff auf eine Datenbank benötigen, die sich an einem schreibgeschützen Speicherort befindet, kann *SetSharedReadOnly (True)* aufgerifen werden, anstatt das *Exclusive* Property zu verwenden. In diesem Modus, erzeugt TurboDB keine net-Dateien, öffnet die Datenbank aber im Shared Mode. Bei Verwendung dieser Methode **muss die Anwendung garantieren, dass keine andere Anwendung die Datenbank modifiziert,** denn TurboDB kann in diesem Modus Änderungen nicht feststellen.

1.2.3 VCL Komponenten Referenz

1.2.3.1 VCL Komponenten

Die folgenden Komponenten sind auf dem TurboDB Reiter in der Delphi Komponenten-Palette zu finden:

TTdbTable

TTdbBatchMove

TTdbDatabase

TTdbQuery (professional edition only)

TTdbEnumValueSet

Die Basisklasse von *TTdbTable* und *TTdbQuery* ist TTdbDataSet, das selbst ein Nachkomme von *TDataSet* ist. Daher verwenden alle Eigenschaften, Methoden und Ereignisse von *TDataSet* die korrespondierenden TurboDB Komponenten.

Weitere TurboDB Klassen:

ETurboDBError

TTdbFieldDef

TTdbFieldDefs

TTdbForeignKeyDef

TTdbForeignKeyDefs

TTdbFulltextIndexDef

TTdbFulltextIndexDefs

Dieses Dokument setzt voraus, dass Sie mit der Delphi-Entwicklungsumgebung vertraut sind und wissen wie Standard Datenzugriffs- und Datensteuerungskomponenten zu verwenden sind.

1.2.3.2 ETurboDBError

Beschreibt einen TurboDB spezifischen Datenbankfehler.

Unit

TdbDataSet

Beschreibung

Wenn TurboDB einen Fehler erkennt, der nicht identisch mit dem korrespondierenden BDE Fehler ist, wird einer *ETurboDBError* Exception anstelle der *EDatabaseError* geworfen. Diese Exception liefert detailliertere Fehlerbeschreibungen.

1.2.3.3 ETurboDBError Hierarchy

Hierarchie

TObject

|

Exception

|

EDatabaseError

|

ETurboDBError

1.2.3.4 ETurboDBError.Properties

In ETurboDBError

TdbError

Reason

Abgeleitet von EDatabaseError

(In der Embarcadero Dokumentation finden Sie weitere Informationen)

HelpContext

Message

1.2.3.5 ETurboDBError.Reason

Beschreibt den Auslöser für einen Fehler.

Delphi Syntax:

```
property Reason: SmallInt;
```

C++ Syntax:

```
__property short Reason = {read=FReason, write=FReason, nodefault};
```

Beschreibung

Prüfen Sie Reason um den reason code eines TurboDB Engine Fehlers zu erhalten.

1.2.3.6 ETurboDBError.TdbError

Gibt die Fehler ID des TurboDB Engine Fehler an.

Delphi Syntax:

```
property TdbError: SmallInt;
```

C++ Syntax:

```
__property short TdbError = {read=FTdbError, write=FTdbError,
nodefault};
```

Beschreibung

Lesen Sie TdbError um den TurboDB Engine Fehler Code zu bestimmen. Die Bedeutung des IDs finden Sie in TurboDB Engine Fehler Codes.

1.2.3.7 TTdbDataSet

Kapselt die TurboDB Funktionalität für Nachkommen von DataSet Objekten.

Unit

TdbDataSet

Beschreibung

TTdbDataSet ist ein Dataset Objekt, das die TurboDB Funktionalität für ein Dataset definiert. Applikationen verwenden niemals ein *TTdbDataSet* Objekts direkt. Stattdessen verwenden sie Nachkommen von *TTdbDataSet*, wie *TTdbTable oder TTdbQuery*, das die Datenbank bezogenen Eigenschaften und Methoden erbt.

Entwickler, die eigene Dataset Komponenten für TurboDB erstellen möchten, werden diese direkt von *TTdbDataSet* ableiten um die gesamt Funktionalität von *TTdbDataSet* und die TurboDB bezogenen Eigenschaften und Methoden zu erben.

1.2.3.8 TTdbDataSet Hierarchy

Hierarchie

TObject
|
TPersistent
|
TComponent
|
TDataSet
|
TTdbDataSet

1.2.3.9 TTdbDataSet Methods

In TTdbDataSet

ActivateSelection

AddToSelection

GetEnumValue

IntersectSelection

IsSelected

Locate

Lookup

RemoveFromSelection

Replace

SaveToFile

Abgeleitet von TDataSet

(In der Embarcadero Dokumentation finden Sie weitere Informationen)

1.2.3.10 TTdbDataSet Properties

In TTdbDataSet

DatabaseName

FieldDefsTdb

Filter

FilterMethod

Filtered

FilterOptions

FilterW

Version

Abgeleitet von TDataSet

(In der Embarcadero Dokumentation finden Sie weitere Informationen)

1.2.3.11 TTdbDataSet Events

In TTdbDataSet

OnProgress

OnResolveLink

Abgeleitet von TDataSet

(In der Embarcadero Dokumentation finden Sie weitere Informationen)

1.2.3.12 TTdbDataSet.ActivateSelection

Aktiviert die aktuelle Selektion

Delphi Syntax:

```
procedure ActivateSelection;
```

C++ Syntax:

```
void __fastcall ActivateSelection(void);
```

Beschreibung

Das Aktivieren der aktuellen Selektion ist wie das Setzen eines Filters: Nur die Datensätze der Selektion werden dann noch angezeigt. *ActivateSelection* setzt das *Filtered* Property auf True, falls es das noch nicht ist.

Siehe auch

Selektionen und Drill-Down

1.2.3.13 TTdbDataSet.AddToSelection

Fügt der Selektion Datensätze hinzu

Delphi Syntax:

```
procedure AddToSelection(RecordNo: TTdbRecordNo); overload;
```

```
procedure AddToSelection(const Filter: UnicodeString); overload;
```

C++ Syntax:

```
void __fastcall AddToSelection(TTdbRecordNo RecordNo);
```

```
void __fastcall AddToSelection(System::String Filter);
```

Beschreibung

Mit der ersten Version wird ein spezifischer Datensatz uir Selektion hinzugefügt. Die zweite Variante fügt diejenigen DAtensätze hinzu, die der Filterbedingung genügen.

Siehe auch

Selektionen und Drill-Down

1.2.3.14 TTdbDataSet.ClearSelection

Entfernt alle Datensätze aus der Selektion

Delphi Syntax:

```
procedure ClearSelection;
```

C++ Syntax:

```
void __fastcall ClearSelection(void);
```

Siehe auch

Selektionen und Drill-Down

1.2.3.15 TTdbDataSet.CreateBlobStream

Liefert ein *TBlobStream*-Objekt zum Lesen oder Schreiben von Daten in das angegebene Blob-Feld.

Delphi Syntax:

```
function CreateBlobStream(Field: TField; Mode: TBlobStreamMode):
TStream;
```

C++ Syntax:

```
virtual Classes::TStream* __fastcall CreateBlobStream(TField* Field,
TBlobStreamMode Mode);
```

Beschreibung

Benutzen Sie *CreateBlobStream* um einen Stream zum Lesen und Schreiben des Wertes für das durch den Parameter Field angegebene Feld zu erhalten. Der Parameter Mode zeigt an, ob der

Stream zum Lesen des Feldwertes (bmRead), Schreiben des Feldwertes (bmWrite), oder Ändern des Feldwertes (bmReadWrite) verwendet wird.

Blob-Streams werden in einem spezifischen Modus für ein spezifisches Feld in einem spezifischen Datensatz erzeugt. Anwendungen müssen einen jedesmal einen neuen Blob-Stream erzeugen, wenn der Datensatz im DataSet wechselt: Benutzen Sie jeden Blob-Stream immer nur einmal.

Hinweis: Sie sollten lieber *CreateBlobStream* aufrufen als den Blob-Stream direkt im Code zu erzeugen. Dies stellt sicher, dass der Stream zum DataSet passt und kann auch gewährleisten, dass bestimmte DataSets die Blob-Daten holen bevor sie den Stream erzeugen.

Hinweis: Bei TurboDB müssen Sie das von *CreateBlobStream* erzeugte Stream-Objekt freigeben, bevor Sie den zugehörigen Datensatz in die Datenbank schreiben. Das ist nötig, weil der Stream erst beim Freigeben seinen Inhalt schreibt.

1.2.3.16 TTdbDataset.DatabaseName

Die Eigenschaft DatabaseName gibt den Namen der Datenbank an, der dieser Datenmenge zugeordnet ist.

Delphi Syntax:

```
property DatabaseName: String;
```

C++ Syntax:

```
__property AnsiString DatabaseName = {read=FDatabaseName, write
=SetDatabaseName};
```

Beschreibung

Mit DatabaseName können Sie den Namen der Datenbank festlegen, die dieser Datenmengenkomponente zugeordnet werden soll. Der Wert dieser Eigenschaft sollte der Name einer Datenbank-Komponente sein, die in der Anwendung verwendet wird. Die können auch ein Verzeichnis wie k:\turbodb\databases\db1 als Datenbanknamen angeben. In diesem Fall sind alle TurboDB Tabellen des Verzeichnisses erreichbar. Eine dritte Variante ist es den Dateinamen einer Single-File Datenbank zuzuweisen.

Hinweis: Der Versuch, dieser Eigenschaft einen Wert zuzuweisen, während eine der Komponente bereits zugeordnete Datenbank geöffnet ist, löst eine Exception aus.

1.2.3.17 TTdbDataSet.FieldDefsTdb

Die Eigenschaft FieldDefs zeigt auf die Liste der Felddefinitionen für die Datenmenge.

Delphi Syntax:

```
property FieldDefsTdb: TTdbFieldDefs;
```

C++ Syntax:

```
__property TTdbFieldDefs* FieldDefsTdb = {read=GetFieldDefsTdb, write
=SetFieldDefsTdb};
```

Beschreibung

FieldDefsTdb ist eine Liste der Felddefinitionen einer Datenmenge auf eine Weise, die speziell für TurboDB ist. *FieldDefsTdb* enthält die Felddefinitionen einer Datenmenge. Sie können diesen Wert in Ihren Anwendungen zwar lesen, um die Felddefinitionen zu ermitteln, sollten die Definitionen aber nur beim Erstellen einer neuen Tabelle mit *CreateTable* ändern.

Wenn Sie Felddefinitionen zu *FieldDefsTdb* hinzufügen, löschen oder ändern, werden die Änderungen in die Standard-*TDataSet.FieldDefs* übernommen und umgekehrt. Verwenden Sie *FieldDefsTdb*, um TurboDB spezifische Felder und Eigenschaften wie Auswahlfelder, Linkfelder oder Relationen einzusetzen. Verwenden Sie *FieldDefs* um mit *TTdbDataSet* auf kompatible Art zu *TDataSet* zu arbeiten.

Um auf die Felder und Feldwerte in einer Datenmenge zuzugreifen, verwenden Sie die Eigenschaften *Fields*, *AggFields* und *FieldValues* sowie die Methode *FieldByName*.

1.2.3.18 TTdbDataSet.Filter

Die Eigenschaft Filter enthält die Filterbedingung der Datenmenge fest.

Delphi Syntax:

```
property Filter: String;
```

C++ Syntax:

```
__property AnsiString Filter = {read=FFilterText, write=SetFilterText};
```

Beschreibung

TurboDB kennt zwei Arten von Filtern, Bedingungs-Filter und Schlüsselwort-Filter.

Bedingungs-Filter arbeiten mit einer logischen Bedingung und zeigen alle Datensätze die dieser Bedingung genügen. Bedingungen sind in Suchbedingungen beschrieben. Mit der Eigenschaft *FilterOptions* wird definiert, welcher Mechanismus verwendet wird.

TurboDB erlaubt als Bedingungs-Filter jede gültige TurboSQL Suchbedingung, nicht mur den limitierten Satz den die BDE-Komponenten bieten. Die Suchbedingung wird in Übereinstimmung zu den Regeln der VCL-Datasets interpretiert:

- Fließkommazahlen könnnen entweder im Stadard-Format, mit Dezimalpunkt oder im aktuellen lokalen Format formattiert werden.
- Zeit-, Datum- und Zeitstempel- (DateTime) Werte beziehen sich auf die lokalen Einstellungen des Systems. Das native Format ohne umschließende Anführungszeichen kann verwendet werden, um eine von den lokalen Einstellungen unabhängiges Darstellung von Zeit-, Datum- und Zeitstempel.-Literal zu erhalten.
- Die Platzhalter-Zeichen für den String-Vergleich mit *like* sind % und _ wie in SQL.

In Delphi 2007 und früher wird *FilterW* gesetzt, um einen Unicode-Filter zu setzen.

Die *TTdbTable* Komponente ermöglicht es auch Schlüsselwort-Fitler zu verwenden, wie sie für einenn Volltext-Index benötigt werden. Schlüsselwort-Filter werden nicht dem *Filter* Property, sondern dem *WordFilter* Property zugewiesen.

Siehe auch

FilterOptions Property
FilterMethod Property

1.2.3.19 TTdbDataSet.Filtered

Die Eigenschaft Filtered gibt an, ob für die Datenmenge ein Filter aktiv ist.

Delphi Syntax:

```
property Filtered: Boolean;
```

C++ Syntax:

```
__property WideString FilterW = {read=FFilterW, write=SetFilterW};
```

Beschreibung

Mit Filtered können Sie ermitteln, ob die Datenmenge gefiltert wird. Hat *Filtered* den Wert True, ist ein Filter aktiviert. Um die mit der Eigenschaft *Filter* oder *FilterW* angegebenen Filterbedingungen oder die Ereignisbehandlungsroutine für *OnFilterRecord* auf die Datenmenge anzuwenden, setzen Sie Filtered auf True.

1.2.3.20 TTdbDataSet.FilterMethod

Spezifiziert die Filter Methode für den folgenden Fitlervorgang.

Delphi Syntax:

```
property FilterMethod: TTdbFilterMethod;
```

C++ Syntax:

```
__property TTdbFilterMethod FilterMethod;
```

Beschreibung

TurboDB unterstützt inkrementelle und statische Filter. Bei inkrementellen Filtern wird der Filter während der Bewegung durch das Dataset angewendet. Bei statischen Filtern wird die Ergebnismenge des Filters im Voraus berechnet und gespeichert, ähnlich wie es bei SQL-Abfragen der Fall ist. In den meisten Fällen sind statische Filter schneller und haben außerdem den Vorteil einer stabilen Ergebnismenge mit zuverlässigen Satznummern, die für Rollbalken etc. verwendet werden können. In einigen speziellen Fällen jedoch, sind inkrementelle Filter ggen sehr große Datenmenge (> 1 Mio Datensätze) deutlich schneller als die statische Variante:

- Falls die Ergebnismenge eine große Untermenge des gesamten Datesets ist, bspw. *Name <> 'Svyczkovski'.*
- Falls die Anwendung eines Index zur Beschleunigung einer Suche nicht möglich ist, bspw. *Date > 2009-10-03 or Amount < 100000.*
- Falls nur der erste Datensatz der geforderten Bedingung genügt.
- Falls eine große Ergebnismenge zusätzlich sortiert werden soll.

Andererseits haben inkrementelle Filter einige Nachteile:

- Die Anzhal der Datensätze kann nicht bestimmt werden.
- Es gibt keine gültige *RecNo* und daher sind keine Rollbalken möglich.
- Sie sind sehr langsam für große Datenmengen und Filter, denen nur wenige Datensätze genügen, bspw. *Name = 'Swyczkovski'*.

FilterMethod muss vor *Filter* gesetzt werden und 2funktioniert nur mit regulären Filtern, nicht mit Wort-Filtern.

1.2.3.21 TTdbDataSet.FilterOptions

Bestimmt ob beim Filtern von Datensätzen partielle Vergleiche erlaubt sind oder nicht.

Delphi Syntax:

```
property FilterOptions: TFilterOptions;
```

C++ Syntax:

```
__property TFilterOptions FilterOptions;
```

Beschreibung

Setzen Sie *FilterOptions* um festzulegen, ob partielle Vergleiche beim Auswerten der Filterbedingung erlaubt sind. Der Wert *foCaseInsensitive* wird ignoriert, da TurboDB mit Kollationen arbeitet, die die Behandlung der Groß-Kleinschreibung definieren.

Wenn ein String in einem Filter mit einem Asterisk (*) endet, kann er zum Vergleich mit Teilstrings verwendet werden. Um dieses Verhalten abzuschalten und den Asterisk als ein literales Zeichen für den String-Vergleich zu verwenden, ist *FilterOptions* der Wert *foNoPartialCompare* zuzuweisen .

1.2.3.22 TTdbDataSet.FilterW

Ein Unicode Ausdruck zur Filterung der Datensätze der Datenmenge.

Delphi Syntax:

```
property FilterW: WideString;
```

C++ Syntax:

```
__property TTdbFieldDefs* FieldDefsTdb = {read=GetFieldDefsTdb, write
=SetFieldDefsTdb};
```

Beschreibung

FilterW wird nur in Delphi 2007 und früher benötigt.

In diesen früheren Delphi-Versionen ist *Filter* ein *AnsiString* und kann nicht-ANSI Zeichen nicht verarbeiten. *FilterW* ist die Unicode Version von Filter. Verwenden Sie *FilterW* um eine Filterbedingung mit nicht-ANSI Zeichen zu definieren. *FilterW* ist nicht published da der Objekt Inspektor Unicode Zeichen nicht unterstützt. *Filter* und *FilterW* hängen voneinander ab. Falls Sie *Filter* einen Wert zuweisen, wird *FilterW* automatisch seinen Wert ändern und umgekehrt. Der Wert von *FilterOptions* bestimmt, wie der Filter angewendet wird.

1.2.3.23 TTdbDataSet.GetEnumValue

Liefert den nummerischen Wert einer Auswahlfeld Konstante.

Delphi Syntax:

```
function GetEnumValue(FieldNo: Integer; const EnumStr: string): Integer;
```

C++ Syntax:

```
int __fastcall GetEnumValue(int FieldNo, const AnsiString EnumStr);
```

Beschreibung

Rufen Sie *GetEnumValue* auf, um den nummerischen Wert zu erhalten der einer bestimmten Konstante eines Auswahlfeldes entspricht. Diese Funktion wird bei TurboDB Tabellen benötigt, die über ein Auswahlfeld verfügen.

1.2.3.24 TTdbDataSet.IntersectSelection

Die Schnittmenge aus der aktuellen Selektion und einem Filter bilden.

Delphi Syntax:

```
procedure IntersectSelection(const Filter: UnicodeString);
```

C++ Syntax:

```
int __fastcall IntersectSelection(const System::String Filter);
```

Beschreibung

IntersectSelection nimmt alle Datensätze aus der aktuellen Selektion, die nicht der Filterbedingung genügen.

Siehe auch

Selektionen und Drill-Down

1.2.3.25 TTdbDataSet.IsSelected

Prüft ob der aktuelle Datensatz in der aktuellen Selektion enthalten ist.

Delphi Syntax:

```
function IsSelected(RecordNo: TTdbRecordNo): Boolean;
```

C++ Syntax:

```
bool __fastcall IsSelected(TTdbRecordNo RecordNo);
```

Beschreibung

IsSelected ist True wenn der aktuelle Datensatz in der aktuellen Selektion enthalten ist.

Siehe auch

Selektionen und Drill-Down

1.2.3.26 TTdbDataSet.Locate

Mit der Methode Locate können Sie einen Datensatz in der Datenmenge suchen und ihn zum aktuellen Datensatz machen.

Delphi Syntax:

```
function Locate(const KeyFields: string; const KeyValues: Variant;
Options: TLocateOptions): Boolean;
```

C++ Syntax:

```
virtual bool __fastcall Locate(const AnsiString KeyFields, const
System::Variant &KeyValues, TLocateOptions Options);
```

Beschreibung

Mit Locate können Sie einen Datensatz in einer Datenmenge suchen und den Cursor auf diesen Datensatz setzen. *KeyFields* ist ein String mit einer durch Semikolons getrennten Liste der Felder, die durchsucht werden sollen. *KeyValues* ist ein variantes Array mit den Werten, die in den Schlüsselfeldern gesucht werden sollen. *Options* ist eine Menge, mit der die Suche in Stringfeldern genauer definiert werden kann. Enthält *Options* das Flag *loCaseInsensitive*, berücksichtigt *Locate* die Groß-/Kleinschreibung nicht. Enthält *Options* das Flag *loPartialKey*, werden teilweise Übereinstimmungen in den Strings von *KeyValues* gesucht. Enthält *Options* eine leere Menge oder handelt es sich bei der Eigenschaft *KeyFields* nicht um Stringfelder, wird *Options* ignoriert. Wird ein übereinstimmender Datensatz gefunden, gibt *Locate* den Wert True zurück und aktiviert den gefundenen Datensatz. Andernfalls wird False zurückgegeben.

1.2.3.27 TTdbDataSet.Lookup

Mit der Methode Lookup können Sie Feldwerte aus einem Datensatz abrufen, der mit bestimmten Suchwerten übereinstimmt.

Delphi Syntax:

```
function Lookup(const KeyFields: String; const KeyValues: Variant; const
ResultFields: String): Variant;
```

C++ Syntax:

```
virtual Variant __fastcall Lookup(const AnsiString KeyFields, const
Variant &KeyValues, const AnsiString ResultFields);
```

Beschreibung

Mit *Lookup* können Sie die Werte der angegebenen Felder aus einem Datensatz abrufen, der die Suchkriterien erfüllt. *KeyFields* ist ein String mit einer durch Semikolons getrennten Liste von Feldnamen, die durchsucht werden sollen.

KeyValues ist ein variantes Array mit den Werten, die mit den Schlüsselfeldern übereinstimmen sollen. Um mehrere Suchwerte festzulegen, übergeben Sie das variante Array *KeyValues* als Argument oder erstellen ein variantes Array mit der Routine *VarArrayOf*.

ResultFields in ein String mit einer durch Semikolons getrennten Liste der Feldnamen, deren Werte aus dem gefundenen Datensatz abgerufen werden sollen.

Lookup gibt ein variantes Array mit den Werten der Felder zurück, die in *ResultFields* angegeben wurden.

1.2.3.28 TTdbDataSet.OnProgress

Das Ereignis OnProgress tritt während einer zeitaufwändigen Operation auf

Delphi Syntax:

```
TTdbProgressEvent = procedure(Sender: TObject; PercentDone: Byte; var
Stop: Boolean) of object;
```

```
property OnProgress: TTdbProgressEvent;
```

C++ Syntax:

```
typedef void __fastcall (__closure *TTdbProgressEvent)(System::TObject*
Sender, Byte PercentDone, bool &Stop);

__property TTdbProgressEvent OnProgress = {read=FOnProgress, write
=FOnProgress};
```

Beschreibung

Verwenden Sie *OnProgress* um eine Fortschrittsanzeige anzuzeigen und dem Anwender eine Möglichkeit zu bieten die Operation abzubrechen. *OnProgress* wird während dem Erzeugen und Ändern von Tabellen und Erzeugen von Indexen aufgerufen.

Hinweis: Führen Sie keine TurboDB Methoden in der Ereignisbehandlungsroutine aus. Da die Datenbank Engine nicht reentrant ist, kann der Aufruf von TurboDB Methoden zu unerwünschten Ergebnissen führen.

1.2.3.29 TTdbDataSet.OnResolveLink

Tritt auf, wenn ein Linkfeld auf einen nicht auflösbaren Wert geändert wurde.

Delphi Syntax:

```
TResolveLinkEvent = procedure(Sender: TObject; FieldNo: Integer; const
LinkInfo: string; var RecordId: Integer; var Cancel: Boolean);

property OnResolveLink: TResolveLinkEvent;
```

C++ Syntax:

```
typedef void __fastcall (__closure *TResolveLinkEvent)(System::TObject*
Sender, int FieldNo, const AnsiString LinkInfo, int &RecordId, bool
&Cancel);

__property TResolveLinkEvent OnResolveLink = {read=FOnResolveLink, write
=FOnResolveLink};
```

Beschreibung

Durch das Schreiben einer Ereignisbehandlungsroutine für *OnResolveLink* haben Sie die Möglichkeit, nach dem Datensatz in der Mastertabelle zu suchen, mit dem eine Verknüpfung erfolgen soll. Die Routine muss in *RecordId* die RecordId des Masterdatensatzes zurückgeben, der mit dem aktuellen Datensatz der Detailtabelle verknüpft werden soll. Um eine Verknüpfung zu verhindern ist *Cancel* auf true zu setzen.

1.2.3.30 TTdbDataSet.RecNo

Gibt den aktuellen Datensatz der Datenmenge an.

Delphi Syntax:

```
property RecNo: Integer;
```

C++ Syntax:

```
__property int RecNo;
```

Beschreibung

Wenn *IsSequenced true* ist, enthält *RecNo* die Sequenznummer des gegenwärtigen Datensatzes in der Datenmenge. Es sollte niemals zur Bearbeitung der Daten verwendet werden sondern ausschließlich für Rollbalken und andere Methoden, dem Benutzer eine Angabe über die aktuelle Position in der Datenmenge zu liefern. Wenn die Anzahl der Datensätze groß ist, kann *RecNo* auch nur eine Schätzung der Position enthalten statt des exakten Wertes.

1.2.3.31 TTdbDataSet.RemoveFromSelection

Entfernt Datensätze aus der aktuellen Selektion.

Delphi Syntax:

```
procedure RemoveFromSelection(RecordNo: TTdbRecordNo); overload;
procedure RemoveFromSelection(const Filter: UnicodeString); overload;
```

C++ Syntax:

```
void __fastcall RemoveFromSelection(TTdbRecordNo RecordNo);
void __fastcall RemoveFromSelection(System::String Filter);
```

Beschreibung

Mit der ersten Variante kann ein spezifischer Datensatz aus der aktuellen Selektion entfernt werden. Mit der zweiten Variante werden alle Datensätze aus der aktuellen Selektion genommen, die der angegebenen Filterbedingung genügen.

Siehe auch

Selektionen und Drill-Down

1.2.3.32 TTdbDataSet.Replace

Ersetzt Feldwerte in mehreren Datensätzen.

Delphi Syntax:

```
procedure Replace(const Filter, Fields, Expressions: string): LongInt;
```

C++ Syntax:

```
int __fastcall Replace(const AnsiString Filter, const AnsiString Fields,
const AnsiString Expressions);
```

Beschreibung

Replace setzt die Inhalte der Felder auf die durch *Expression* berechneten Werte. Und zwar für alle Datensätze, die der in *Filter* definierten Bedingung genügen. Der Ausdruck wir für den Kontext jedes Datensatzes ausgewertet.

Beispiel

Das folgende Beispiel fügt die Zahl Neun an die Telefonnummern der Stadt Reading in England:

```
Replace('Phone like "+44 118*", 'Phone', 'Phone + "9"');
```

1.2.3.33 TTdbDataSet.SaveToFile

Schreibt das gesamte DataSet in eine Datei.

Delphi Syntax:

```
function SaveToFile(const FileName: string; Format: TTdbTableFormat):
Integer;
```

C++ Syntax:

```
int __fastcall SaveToFile(const AnsiString FileName, Tdbtypes::
TTdbTableFormat Format);
```

Beschreibung

Verwenden Sie *SaveToFile* wenn Sie eine Kopie Ihres DataSets machen wollen. *SaveToFile* verwendet intern die *TTdbBatchMove* Komponente um eine Datei mit allen Daten in einem der verfügbaren Formate zu erstellen. *Format* definiert dabei den Dateityp.

Die Methode ist besonders hilfreich, wenn Sie eine Ergebnismenge einer Query zur weiteren Bearbeitung abspeichern möchten. Sie könnten die das Result Set z.B. im TurboDB Format abspeichern und es anschließend mit einer TTdbTable Komponente öffnen oder eine weitere

Query darauf ausführen.

1.2.3.34 TTdbDataSet.Version

Gibt die Version der TurboDB Komponenten und der TurboDB Engine an.

Delphi Syntax:

```
property Version: String;
```

C++ Syntax:

```
__property AnsiString Version = {read=GetVersion, write=SetVersion};
```

Beschreibung

Version ist eine Zeichenfolge wie 2.0.34/4.1.1. Der Teil vor dem Schrägstrich ist die Version der TurboDB Komponenten, der Teil dahinter steht für die TurboDB Engine. Beide Versionsnummern haben eine Haupt- und eine Unterversionsnummer und eine Buildnummer.

Hinweis: Die Eigenschaft kann nur gelesen werden.

1.2.3.35 TTdbForeignKeyAction

Bezeichnet auf welche Art eine Tabelle geschützt oder verschlüsselt wird.

Unit

TdbDataSet

Delphi Syntax:

```
type TTdbForeignKeyAction = (tiaReject, tiaSetNull, tiaSetDefault,
tiaCascade);
```

C++ Syntax:

```
enum TTdbForeignKeyAction {tiaReject, tiaSetNull, tiaSetDefault,
tiaCascade};
```

Beschreibung

Die Werte dieses Typs beschreiben, wie TurboDB auf die Verletzung eines Fremdschlüssels reagiert.

Wert	Beschreibung
tiaReject	Die Änderung, die zur Verletzung führt, wird nicht durchgeführt.
tiaSetNull	Die den Fremdschlüssel beschreibenden Felder der Kind-Tabelle sind null. Noch nicht implementiert.
tiaSetDefault	Die Fremdschlüssel-Felder der Kind-Tabelle werden auf Vorgabewerte gesetzt. Noch nicht implementiert.
tiaCascade	Die korrespondierenden Datensätze in der Kind-Tabelle werden gelöscht (wenn der Elterndatensatz gelöscht wird) oder geändert (wenn der Elterndatensatz geändert wird).

1.2.3.36 TTdbForeignKeyDef

TTdbForeignKeyDef wird verwendet, um Beziehungen von einer Tabelle zu anderen über Fremdschlüssel zu definieren und anzuzeigen.

Unit

TdbDataSet

Beschreibung

TTdbForeignKeyDef definiert über korrespondierende Felder eine Beziehung zwischen einer Kind- und einer Eltern-Tabelle.

1.2.3.37 TTdbForeignKeyDef Hierarchy

Hierarchie

TObject

|

TPersistent

|

TCollectionItem

|

TTdbForeignKeyDef

1.2.3.38 TTdbForeignKeyDef Methods

In TTdbForeignKeyDef

Assign

Abgeleitet von TCollectionItem

(weitere Informationen in der Embarcadero Dokumentation)

1.2.3.39 TTdbForeignKeyDef Properties

In TTdbTable

ChildFields

DeleteAction

Name

ParentTableName

ParentFields

UpdateAction

Abgeleitet von TCollectionItem

(weitere Informationen in der Embarcadero Dokumentation)

1.2.3.40 TTdbForeignKeyDef.Assign

Kopiert die Eigenschaften einer Fremdschlüsseldefinition in eine andere.

Delphi Syntax

```
procedure Assign(Source: TPersistent); override;
```

C++ Syntax

```
virtual void __fastcall Assign(Classes::TPersistent* Source);
```

Beschreibung

Unterstützt den Standard-VCL Mechanismus um zu kopieren.

1.2.3.41 TTdbForeignKeyDef.ChildFields

Enthält einer Liste von Feldnamen der Eltern-Tabelle.

Delphi Syntax:

```
property ChildFields: string;
```

C++ Syntax:

```
__property AnsiString ChildFields = {read=GetChildFields, write
=SetChildFields};
```

Beschreibung

Die Werte der Felder, die hier angegeben werden, müssen den Werten der *Parent fields* in der *Parent table* entsprechen. Trennen Sie die Spaltennamen durch Semikolon.

1.2.3.42 TTdbForeignKeyDef.DeleteAction

Legt die Reaktion der Datenbank Engine fest, falls der Eltern-Datensatz eines Kind-Datensatzes gelöscht wird.

Delphi Syntax:

```
property DeleteAction: TTdbForeignKeyAction;
```

C++ Syntax:

```
__property TTdbForeignKeyAction DeleteAction = {read=GetDeleteAction,
write=SetDeleteAction};
```

Beschreibung

Falls ein Datensatz der Eltern-Tabelle gelöscht wird, wird die definierte Aktion für alle Datensätze der Kind-Tabelle ausgeführt, die mit dem Eltern-Datensatz verknüpft sind.

Siehe auch

TTdbForeignKeyAction

1.2.3.43 TTdbForeignKeyDef.Name

Bezeichnet den Namen der Fremdschlüsseldefinition.

Delphi Syntax:

```
property Name: string
```

C++ Syntax:

```
__property AnsiString Name = {read=GetName, write=SetName};
```

Beschreibung

Der Name identifiziert die Definition eines Fremdschlüssel. dieser kann auch in SQL Statements verwendet werden.

1.2.3.44 TTdbForeignKeyDef.ParentTableName

Bezeichnet die Eltern-Tabelle für diesen Fremdschlüssel.

Delphi Syntax:

```
property ParentTableName: string;
```

C++ Syntax:

```
__property AnsiString ParentTableName = {read=GetParentTableName, write
=SetParentTableName};
```

Beschreibung

ParentTableName muss der Name einer anderen Tabelle in derselben Datenbank sein, zu der die Kind-Tabelle gehört. Die Werte der Child fields jedes Datensatzes der Kind-Tabelle müssen zu den Werten der Parent fields eines Datensatzes in der Eltern-Tabelle korrespondieren.

1.2.3.45 TTdbForeignKeyDef.ParentFields

Beschreibt die Liste der Felder in der Eltern-Tabelle.

Delphi Syntax:

```
property ParentFields: string;
```

C++ Syntax:

```
__property AnsiString ParentFields = {read=GetParentFields, write=Set
ParentFields};
```

Beschreibung

Es muss die gleiche Anzahl an Spaltennamen (getrennt durch Semikolon) in dieser Eigenschaft gegeben sein wie in der ChildFields Eigenschaft. Für jeden Datensatz der Kind-Tabelle muss ein Datensatz in der Eltern-Tabelle existieren, wobei die Werte dieser Felder mit den Werten der Felder in der Kind-Tabelle übereinstimmen.

1.2.3.46 TTdbForeignKeyDef.UpdateAction

Definiert die Reaktion der Datenbank Engine, falls ein Eltern-Datensatz geändert wird.

Delphi Syntax:

```
property UpdateAction: TTdbForeignKeyAction;
```

C++ Syntax:

```
__property TTdbForeignKeyAction UpdateAction = {read=GetUpdateAction,
write=SetUpdateAction};
```

Beschreibung

Falls ein Datensatz der Eltern-Tabelle in einer Art geändert wird, dass er nicht mehr zu den Werten der korrespondierenden Kind-Datensätze passt, wird die hier definierte Aktion auf alle diese Kind-Datensätze ausgeführt.

Siehe auch

TTdbForeignKeyAction

1.2.3.47 TTdbForeignKeyDefs

TTdbForeignKeyDefs enthält alle Fremdschlüssel einer Tabelle.

Unit

TdbDataSet

Description

TTdbForeignKeyDefs enthält *TTdbForeignKeyDef* Objekte, die zu einer Tabelle gehören. Die Property *TTdbTable.ForeignKeyDefs* enthält ein *TTdbForeignKeyDefs* Objekt.

1.2.3.48 TTdbForeignKeyDefs Hierarchy

Hierarchy

TObject

|

TPersistent

|

TCollection

|

TOwnedCollection

|

TDefCollection

|

TTdbForeignKeyDefs

1.2.3.49 TTdbForeignKeyDefs Methods

In *TTdbForeignKeyDefs*

Add

Abgeleitet von *TDefCollection*

(weitere Informationen in der Embarcadero Dokumentation)

1.2.3.50 TTdbForeignKeyDefs.Add

Erzeugt einen neuen Fremdschlüssel und fügt diesen der *TTdbForeignKeyDefs* Liste hinzu.

Delphi Syntax

```
function Add(ParentTableName, ParentFields, ChildFields: string;
UpdateAction: TTdbForeignKeyAction = tiaReject; DeleteAction:
TTdbForeignKeyAction = tiaReject): TTdbForeignKeyDef; overload;
```

C++ Syntax

```
virtual TTdbForeignKeyDef* __fastcall Add(const System::String
ParentTableName, const System::String ParentFields, const System::String
ChildFields, TTdbForeignKeyAction UpdateAction = tiaReject,
TTdbForeignKeyAction DeleteAction = tiaReject);
```

Beschreibung

Verwenden Sie *Add* um für eine Tabelle einen Fremdschlüssel in einem einzigen Funktionsaufruf zu erstellen.

1.2.3.51 TTdbFulltextIndexDef

TTdbFulltextIndexDef beschreibt einen Volltext-Index in einer Datenbank-Tabelle.

Unit

TdbDataSet

Beschreibung

Verwenden Sie die Eigenschaften und Methoden einer Volltext-Index Beschreibung um:

- Einen Volltext-Index für eine Tabelle zu erstellen
- Existierende Volltext-Index anzuzeigen und zu überprüfen.
- Die Felder, die in den Volltext-Index eingehen festzulegen.
- Den Namen eines Volltext-Index zu bestimmen.

TTdbFulltextIndexDef wird nur für den neuen Volltext-Index Typ verwendet, der ab Tabellen-Level 4 und höher verfügbar ist.

1.2.3.52 TTdbFulltextIndexDef Hierarchy

Hierarchie

TObject

|

TPersistent

|

TCollectionItem

|

TNamedItem

|

TTdbFulltextIndexDef

1.2.3.53 TTdbFulltextIndexDef Methods

In TTdbForeignKeyDef

Assign

Abgeleitet von TCollectionItem

(weitere Informationen in der Embarcadero Dokumentation)

1.2.3.54 TTdbFulltextIndexDef Properties

In TTdbTable

Dictionary

Fields

MinRelevance

Options

Abgeleitet von TNamedItem

(weitere Informationen in der Embarcadero Dokumentation)

1.2.3.55 TTdbFulltextIndexDef.Assign

Kopiert die Eigenschaften einer Volltext-Index Definition in eine andere.

Delphi Syntax

```
procedure Assign(Source: TPersistent); override;
```

C++ Syntax

```
virtual void __fastcall Assign(Classes::TPersistent* Source);
```

Beschreibung

Unterstützt den VCL Standard-Kopier-Mechanismus.

1.2.3.56 TTdbFulltextIndexDef.Dictionary

Beschreibt den Namen der Wörterbuchtabelle für den Volltext-Index.

Delphi Syntax:

```
property Dictionary: string;
```

C++ Syntax:

```
__property AnsiString Dictionary = {read=GetDictionary, write
=SetDictionary};
```

Beschreibung

Ein Volltext-Index bezieht sich immer auf eine Wörterbuchtabelle, die die indizierten Wörter aufnimmt. Diese Tabelle muss existieren bevor der Volltext-Index erstellt wird.

Siehe auch

Einen Volltextindex zur Entwurfszeit erzeugen, Einen Volltextindex zur Laufzeit erzeugen

1.2.3.57 TTdbFulltextIndexDef.Fields

Beschreibt die Felder, die in den Volltext-Index eingehen sollen.

Delphi Syntax:

```
property Fields: string;
```

C++ Syntax:

```
__property AnsiString Fields = {read=GetFields, write=SetFields};
```

Beschreibung

Fields ist eine mit Liste von Feldnamen (mit Bindestrichen getrennt), die in den Volltext-Index eingehen sollen.

1.2.3.58 TTdbFulltextIndexDef.MinRelevance

Definiert das Minimum der Relevanz für die zu indizierenden Wörter.

Delphi Syntax:

```
property MinRelevance: SmallInt;
```

C++ Syntax:

```
__property short MinRelevance = {read=GetMinRelevance, write
=SetMinRelevance};
```

Beschreibung

Wörter mit einer geringeren Relevanz als der hier festgelegten werden nicht in den Volltext-Index aufgenommen, nach diesen Wörtern kann auch nicht gesucht werden. *MinRelevance* ist eine positive Zahl zwischen 0 (absolut irrelevant) und 100 (sehr relevant). Relevanz-Unterstützung nicht vollständig implementiert.

1.2.3.59 TTdbFulltextIndexDef.Options

Beschreibt die Charakteristik eines Volltext-Index.

Delphi Syntax

```
property Options: TFulltextIndexOptions;
```

C++ Syntax:

```
__property TFulltextIndexOptions Options = {read=FOptions, write
=SetOptions, nodefault}
```

Beschreibung

Beim Erstellen eines neuen Volltext-Index, wird *Options* verwendet um die Attribute des Index festzulegen. *Options* kann null oder mehrere *TFulltextIndexOption* Konstanten beinhalten.

Beim Untersuchen eines Volltext-Index kann *Options* gelesen werden, um herauszufinden welche Eigenschaften verwendet wurden um den Volltext-Index anzulegen.

1.2.3.60 TTdbFulltextIndexOptions

TTdbFulltextIndexOptions beschreibt die Attribute eines Volltext-Index.

Unit

TdbDataSet

Delphi Syntax:

```
type
  TTdbFulltextIndexOption = (tfoUpdateDictionary);
  TTdbFulltextIndexOptions = set of TTdbFulltextIndexOption;
```

C++ Syntax:

```
enum TFulltextIndexOption {tfoUpdateDictionary};
typedef Set<TIndexOption, tfoUpdateDictionary, tfoUpdateDictionary>
TTdbFulltextIndexOptions;
```

Beschreibung

TTdbFulltextIndexOptions ist eine Menge an Attributen, die sich auf einen bestimmten Volltext-Index beziehen. Ein *TTdbFulltextIndexOptions* Wert kann null oder mehrere der folgenden Werte beinhalten:

Wert	Beschreibung
`tfoUpdateDictionary`	Während der Indizierung wird TurboDB nur Wörter in die Wörterbuchtabelle aufnehmen, die in der Tabelle, aber nicht in der Wörterbuchtabelle vorhanden sind.

1.2.3.61 TTdbTable

TTdbTable kapselt eine TurboDB Datenbanktabelle.

Unit

TdbDataSet

Beschreibung

Verwenden Sie *TTdbTable* um auf die Daten einer einzelnen TurboDB Tabelle zuzugreifen. *TTdbTable* ermöglicht den direkten Zugriff auf jeden einzelnen Datensatz der Tabelle. Durch die Verwendung von Filtern kann eine Tabellen-Komponente auch auf einer Untermenge der vorhandenen Datensätze arbeiten.

1.2.3.62 TTdbTable Hierarchy

Hierarchie

TObject

|

TPersistent

|

TComponent

|

TDataSet

|

TTdbDataSet

|

TTdbTable

1.2.3.63 TTdbTable Methods

In TTdbTable

AddIndex
AlterTable
BatchMove
CreateTable
DeleteIndex
DeleteTable
EditKey
EmptyTable
Exists
FindKey
FindNearest
GetUsage
GotoKey
GotoNearest
LockTable
RenameTable
SetKey
UnlockTable
UpdateFullTextIndex
UpdateIndex

Abgeleitet von TTdbDataSet
GetEnumValue
Locate
Lookup
Replace

Abgeleitet von TDataSet

(In der Embarcadero Dokumentation finden Sie weitere Informationen)

1.2.3.64 TTdbTable Properties

In TTdbTable

DetailFields
Exclusive
FlushMode
FullTextTable
IndexDefs

IndexName
Key
LangDriver
MasterSource
Password
ReadOnly
TableFileName
TableLevel
TableName

Abgeleitet von TTdbDataSet

DatabaseName
FieldDefsTdb
Filter
Filtered
FullTextTable
Version

Abgeleitet von TDataSet

(In der Embarcadero Dokumentation finden Sie weitere Informationen)

1.2.3.65 TTdbTable Events

Abgeleitet von TTdbDataSet

OnProgress
OnResolveLink

Abgeleitet von TDataset

(In der Embarcadero Dokumentation finden Sie weitere Informationen)

1.2.3.66 TTdbTable.AddFulltextIndex

Erstellt einen neuen Volltext-Index für die Tabelle.

Delphi Syntax:

```
procedure AddFulltextIndex(const Fields, RelationField,
CounterIndexFileName: string; Limit: Integer);
```

C++ Syntax:

```
void __fastcall AddFulltextIndex(const AnsiString Fields, const
AnsiString RelationField, const AnsiString CounterIndexFileName, int
Limit);
```

Beschreibung

AddFulltextIndex wird verwendet um einen neuen Volltext-Index für die Tabelle zu erstellen. Die

Methode erzeugt die klassische Form eines nicht gewarteten Volltext-Index.

Fields ist die Liste der Spaltennamen die in den Volltext-Index aufgenommen werden. *RelationField* ist das Relations-Feld der Tabelle, das die Verbindung zwischen der Stichwort-Tabelle und der Tabelle herstellt. *CounterIndexFileName* ist der Name einer Text-Datei mit Wörtern, die nicht in den Index aufgenommen werden sollen. *Limit* ist die Anzahl die beschreibt wie oft ein Wort auftreten darf bevor es aus dem Index entfernt wird.

Anmerkung

Es wird empfohlen ein Upgrade der Tabelle auf Tabellen-Level 4 durchzuführen und AddFulltextIndex2 zu verwenden um einen gewarteten Volltext-Index zu verwenden.

See also

AddFulltextIndex2

1.2.3.67 TTdbTable.AddFulltextIndex2

Erstellt einen Volltextindex der 2ten Generation für die Tabelle.

Delphi Syntax:

```
procedure TTdbTable.AddFulltextIndex2(
     const Name, Fields, Dictionary: AnsiString
     const Separators: UnicodeString = ''
);
```

C++ Syntax:

```
void __fastcall AddFulltextIndex2(
     const AnsiString Name,
     const AnsiString Fields,
     const AnsiString Dictionary,
     const UnicodeString Separators = ""
);
```

Beschreibung

Die Methode ist ab Tabellen-Level 4 verfügbar. Sie erstellt einen neuen, gewarteten Volltext-Index für die Tabelle. *Name* ist der Name des Index, *Fields* ist eine Komma-separierte Liste der Felder, die in den Index aufgenommen werden und *Dictionary* ist der Name der Volltext-Index Tabelle, die alle Wörter enthält.

Separators ist eine Liste der Zeichen, die als Trennzeichen dienen sollen, wenn die Texte der Tabelle oder eine Suchbedingung in einzelne Wörter aufgespalten werden. Es kann auch ein leerer String übergeben werden. In diesem Fall werden die Voreinstellungen verwendet:

```
 °^!?'"§$%&/\()[]{}<>=´`+-*~'#,.;:@|
```

Falls ein nicht-leerer String übergeben wird, werden alle Zeichen mit einer Kodierung < 32 plus die übergebenen Zeichen als Trennzeichen betrachtet.

Kompatibilität

Benutzerdefiierte Trennzeichen sind ab Tabellen-Level 6 verfügbar. Der Versuch sie bei einem niedrigerem Tabellen-Level zuzuweisen, wirft eine "Unsupported table feature" Exception.

Siehe auch

CREATE FULLTEXTINDEX TurboSQL Kommando

1.2.3.68 TTdbTable.AddIndex

Die Methode AddIndex erzeugt einen neuen Index für die Tabelle.

Delphi Syntax:

```
procedure AddIndex(const Name, Fields: String; Options: TIndexOptions,
const DescFields: String = '');
```

C++ Syntax:

```
void __fastcall AddIndex(const AnsiString Name, const AnsiString Fields,
Db::TIndexOptions Options, const AnsiString DescFields = "");
```

Beschreibung

Rufen Sie *AddIndex* auf um einen neuen Indes für die Tabelle zu erstellen. Ein Index wird benötigt um die Datensätze der Tabelle sortiert anzuzeigen oder um die Suche nach bestimmten Datensätzen zu beschleunigen.

Name ist der Name des neuen Index, z.B. MyTableByName. Der Dateiname des neuen Index wird dann MyTableByName.ind lauten.

Fields ist die Liste der Tabellenspalten, die für den Index verwendet werden soll, separiert durch Semikolons.

Options ist ein Set von Indexoptionen. Die verfügbaren Optionen sind:

ixExpression Die Eigenschaft *Fields* beinhaltet einen TurboDB Ausdruck um die Indexeinträge zu berechnen

ixUnique Jeder Indexeintrag darf nur einmal vorkommen

DescFields ist eine Liste der Felder in *Fields*, die in absteigender Reihenfolge sortiert werden.

Anmerkung

Die Methode ist äquivalent zu *TTable.AddIndex*.

1.2.3.69 TTdbTable.AlterTable

Restrukturiert die Datenbanktabelle.

Delphi Syntax:

```
procedure AlterTable;
```

C++ Syntax:

```
void __fastcall AlterTable(void);
```

Beschreibung

Verwenden Sie *AlterTable* um die Spalten, den Tabellenlevel, das Passwort oder den Schlüssel einer Datenbanktabelle zu ändern. Setzen Sie vor einem Aufruf von *AlterTable* die Eigenschaften *FieldDefsTdb*, *TableLevel*, *Password* und *Key* auf die gewünschten Werte. Falls das Ereignis *OnProgress* mit einer Routine verknüpft ist, benachrichtigt *AlterTable* die Ereignisbehandlungsroutine während der Aktion.

Hinweis

Rufen Sie in der Ereignisbehandlungsroutine für OnProgress keine TurboDB Methoden auf, das dies die TurboDB Engine in einen undefinierten Zustand versetzen kann.

1.2.3.70 TTdbTable.BatchMove

Die Methode *BatchMove* verschiebt Datensätze aus einer Datenmenge in die Tabelle.

Delphi Syntax:

```
function BatchMove(ASource: TDataSet; AMode: TTdbBatchMode): LongInt;
```

C++ Syntax:

```
int __fastcall BatchMove(Db::TDataSet* ASource, Tdbtypes::TTdbBatchMode
AMode);
```

Beschreibung

Rufen Sie BatchMove auf, um folgende Operationen durchzuführen:

- Kopieren von Datensätzen aus einer anderen in diese Tabelle.
- Aktualisieren von Datensätzen in dieser Tabelle, die auch in einer anderen Tabelle

existieren.
- Datensätze aus einer anderen Tabelle am Ende dieser Tabelle anfügen.
- Löschen von Datensätzen aus dieser Tabelle, die in einer anderen Tabelle existieren.

ASource ist eine Datenmengen-Komponente, die Datensätze zum Import (oder beim Löschen) für die Übereinstimmung in dieser Tabelle enthält. Der Parameter AMode gibt an, welche Operation (Kopieren, Aktualisieren, Anfügen oder Löschen) ausgeführt werden soll. Diese Tabelle ist das Ziel der Batch-Operation.

BatchMove gibt die Anzahl der bearbeiteten Datensätze zurück.

1.2.3.71 TTdbTable.Capacity

Legt die Mindestanzahl der Datensätze fest, die die Tabelle aufnehmen muss.

Delphi Syntax:

```
property Capacity: LongInt;
```

C++ Syntax:

```
__property long Capacity = {read, write, nodefault};
```

Beschreibung

Während alle TurboDB-Tabellen bis zu zwei Milliarden Datensätze aufnehmen können, hängt die Anzahl der maximal möglichen Indexeinträge von der Größe der zu indizierenden Felder und von der Seitengröße ab.Setzen Sie diese Eigenschaft bevor Sie die Tabelle erzeugen oder ändern, um sicherzustellen daß Indexe mit den richtigen Einstellungen erzeugt werden. Es ist nicht notwendig diese Eigenschaft auf einen exakten Wert zu setzen, ein Näherungswert der zu erwartenden Größe reicht aus. Wenn Sie beispielsweise eine Tabellengröße von 100.000 Datensätzen erwarten, setzen Sie diese Eigenschaft auf 200.000. Die Indexe haben den angegebenen Wert als Mindestkapazität, können aber meistens wesentlich größer werden. (Die tatsächliche Größe eines Index können Sie im TurboDB Viewer sehen.) Der Zweck dieses Wertes liegt also hauptsächlich in der Optimierung.

Hinweis

Diese Eigenschaft kann nur für Tabellen mit Tabellen-Level 4 oder höher gesetzt werden. Wenn Sie Indexe für Tabellen mit älteren Formaten erzeugen wird die Indexkapazität anhand des Wertes der Eigenschaft IndexCapacity der Database Komponente errechnet.

1.2.3.72 TTdbTable.Collation

Die Kollation der Tabelle

Delphi Syntax:

```
property Collation: string;
```

C++ Syntax:

```
__property AnsiString Collation;
```

Beschreibung

Die Kollation einer Tabelle ist die Standard-Kollation für die textuellen Spalten der Tabelle. (siehe *TTdbFieldDef.Specification*). Diese Eigenschaft wird beim Erzeugen und Restrukturieren der Tabelle verwendet. Wir ein leerer String angegeben, wird die Kollation *TurboDB* verwendet.

Collation ersetzt das *LangDriver* Property, das bis TurboDB 5 verwendet wurde.

Siehe auch

Kollationen
TTdbFieldDef.Specification

1.2.3.73 TTdbTable.CreateTable

Die Methode CreateTable erzeugt eine neue Tabelle.

Delphi Syntax:

```
procedure CreateTable;
```

C++ Syntax:

```
void __fastcall CreateTable(void);
```

Beschreibung

Mit dem Aufruf von CreateTable kann zur Laufzeit eine Tabelle erzeugt werden. Setzen Sie vor dem Aufruf die Eigenschaften FieldDefsTdb, DatabaseName, TableName, TableLevel, Password und Key auf die gewünschten Werte. Falls das Ereignis *OnProgress* mit einer Routine verknüpft ist, benachrichtigt *AlterTable* die Ereignisbehandlungsroutine während der Aktion.

Hinweis

Rufen Sie in der Ereignisbehandlungsroutine für *OnProgress* keine TurboDB Methoden auf, das dies die TurboDB Engine in einen undefinierten Zustand versetzen kann.

1.2.3.74 TTdbTable.DeleteAll

Löscht alle Datensätze im aktuellen Filter-Bereich.

Delphi Syntax:

```
procedure DeleteAll;
```

C++ Syntax:

```
void __fastcall DeleteAll(void);
```

Beschreibung

Mit *DeleteAll* können Sie viele Datensätze auf einmal löschen. Falls ein Filter gesetzt ist, werden alle gefilterten Datensätze gelöscht. Falls kein Filter gelöscht ist, werden alle Datensätze der Tabelle gelöscht. In diesem Fall arbeitet DeleteAll ähnlich wie *EmptyTable*, es ist aber nicht so schnell und benötigt dafür keinen exklusiven Zugriff auf die Tabelle.

1.2.3.75 TTdbTable.DeleteIndex

Die Methode DeleteIndex löscht einen Index der Tabelle.

Delphi Syntax:

```
procedure DeleteIndex(const Name: string);
```

C++ Syntax:

```
void __fastcall DeleteIndex(const AnsiString Name);
```

Beschreibung

Verwenden Sie *DeleteIndex* um einen Index der Tabelle zu entfernen und die Indexdatei zu löschen.

Name ist der Dateiname des Index inklusive Erweiterung. Er ist identisch mit dem korrespondierenden Eintrag in *IndexFiles*.

Anmerkung

TTdbTable.DeleteIndex ist Äquivalent zu *TTable.DeleteIndex*.

1.2.3.76 TTdbTable.DeleteTable

Die Methode *DeleteTable* löscht eine vorhandene Datenbanktabelle.

Delphi Syntax:

```
procedure DeleteTable;
```

C++ Syntax:

```
void __fastcall DeleteTable(void);
```

Beschreibung

Verwenden Sie *DeleteTable* um eine Datenbanktabelle und alle zugehörigen Dateien zu löschen.

Hinweis

Die Daten der Tabelle können anschließend nicht mehr wiederhergestellt werden.

1.2.3.77 TTdbTable.DetailFields

Definiert die Feldnamen die zur Verknüpfung der Tabelle mir einer Masterdatenquelle verwendet werden.

Delphi Syntax:

```
property DetailFields: string;
```

C++ Syntax:

```
__property AnsiString DetailFields = {read=FDetailFields, write
=SetDetailFields};
```

Beschreibung

Setzen Sie *DetailFields* um die Felder zu bestimmen deren Werte mit den korrespondierenden Felder der Mastertabelle übereinstimmen müssen. *DetailFields* ist eine Zeichenkette, die einen oder mehrere Feldnamen der Detailtabelle beinhaltet. Die einzelnen Felder sind durch Semikolons zu trennen. Wenn Sie *DetailFields* verwenden, müssen Sie auch die Eigenschaft MasterFields auf die korrespondierenden Feldnamen der Mastertabelle setzen. Falls Sie *MasterFields* verwenden, *DetailFields* dagegen leer lassen, geht TurboDB davon aus, dass die Feldnamen der Detailtabelle mit denen der Mastertabelle übereinstimmen.

1.2.3.78 TTdbTable.EditKey

Mit der Methode *EditKey* kann der Inhalt des Suchschlüsselpuffers geändert werden.

Delphi Syntax:

```
procedure EditKey;
```

C++ Syntax:

```
void __fastcall EditKey(void);
```

Beschreibung

Durch den Aufruf von EditKey kann der Datenmenge der Status dsSetKey zugewiesen werden, während der Inhalt des aktuellen Suchschlüsselpuffers erhalten bleibt. Um die aktuellen Suchschlüssel zu ermitteln, können Sie die Eigenschaft IndexFields verwenden, um über alle Felder zu iterieren, die im aktuellen Index enthalten sind.

EditKey ist hilfreich, wenn mehrere Suchvorgänge durchgeführt und zwischen den Suchvorgängen nur einer oder zwei der Feldwerte geändert werden.

1.2.3.79 TTdbTable.EmptyTable

Die Methode *EmptyTable* löscht alle Datensätze aus der Tabelle.

Delphi Syntax:

```
procedure EmptyTable;
```

C++ Syntax:

```
void __fastcall EmptyTable(void);
```

Beschreibung

Die Methode EmptyTable löscht sämtliche Datensätze aus der Datenbanktabelle.

Hinweis

Die gelöschten Datensätze können nicht wiederhergestellt werden.

Anmerkung

TTdbTable.EmptyTable ist äquivalent zu *TTable.EmptyTable*.

1.2.3.80 TTdbTable.EncryptionMethod

Definiert ob und auf welche Art eine Tabelle geschützt und verschlüsselt ist.

Delphi Syntax:

```
procedure EncryptionMethod: TTdbEncryptionMethod
```

C++ Syntax:

```
__property TTdbEncryptionMethod EncryptionMethod = {read, write,
nodefault};
```

Beschreibung

Eine TurboDB Tabelle kann auf unterschiedliche Art geschützt und verschlüsselt werden. Diese Eigenschaft zeigt an welche Art gewählt wurde und wird verwendet um bei der Erzeugung einer Tabelle mit *CreateTable* oder *AlterTable* die Methode der Verschlüsselung festzulegen.

See also

TTdbEncryptionMethod, Password, Data Security

1.2.3.81 TTdbTable.Exclusive

Die Eigenschaft Exclusive ermöglicht es einer Anwendung, exklusiven Zugriff auf eine TurboDB-Tabelle zu erhalten.

Delphi Syntax:

```
property Exclusive: Boolean;
```

C++ Syntax:

```
__property bool Exclusive = {read=FExclusive, write=SetExclusive,
nodefault};
```

Beschreibung

Mit *Exclusive* kann der Zugriff auf eine TurboDB-Tabelle durch andere Anwendungen verhindert werden, während diese in der aktuellen Anwendung geöffnet ist. Weisen Sie der Eigenschaft Exclusive vor dem Öffnen der Tabelle den Wert True zu.

Wenn *Exclusive* den Wert True hat, kann nach dem erfolgreichen Öffnen der Tabelle keine andere Anwendung mehr darauf zugreifen. Wenn sich die Tabelle bereits im Zugriff einer anderen Anwendung befindet, wird eine Exception ausgelöst. Für diese Exception muß der Code, der die Tabelle öffnet in einen try..except-Block eingeschlossen werden.

Weisen Sie *Exclusive* während des Entwurfs nicht den Wert True zu, wenn Sie beabsichtigen, der Eigenschaft Active ebenfalls den Wert True zuzuweisen. In diesem Fall wird eine Exception ausgelöst, da die Tabelle bereits von der IDE geöffnet wurde.

Achtung

Exklusiver Zugriff auf einer Tabelle innerhalb einer SingleFile Datenbank ist nicht möglich. Beim Versuch wird eine Exception ausgelöst.

Öffnen Sie stattdessen die gesamte Datenbank im exklusiven Modus.

1.2.3.82 TTdbTable.Exists

Zeigt an ob die zugrundeliegende Datenbanktabelle existiert.

Delphi Syntax:

```
property Exists: Boolean;
```

C++ Syntax:

```
__property bool Exists = {read=GetExists, nodefault};
```

Beschreibung

Lesen Sie *Exists* zur Laufzeit um zu bestimmen, ob die Datenbanktabelle existiert. Falls die Tabelle nicht existiert, können Sie die Tabelle mit *CreateTable* erzeugen, wobei die Feld- und Indexdefinitionen berücksichtigt werden.

Die Eigenschaft kann nur gelesen werden.

1.2.3.83 TTdbTable.FindKey

Die Methode FindKey sucht nach einem Datensatz, der die angegebenen Feldwerte enthält.

Delphi Syntax:

```
function FindKey(const KeyValues: array of const): Boolean;
```

C++ Syntax:

```
bool __fastcall FindKey(const System::TVarRec* KeyValues, const int
KeyValues_Size);
```

Beschreibung

Mit *FindKey* können Sie nach einem bestimmten Datensatz in einer Datenmenge suchen. *KeyValues* enthält ein Komma-Getrenntes-Array mit Feldwerten, das auch als Schlüssel bezeichnet wird. Jeder Wert des Schlüssels kann ein Literal, eine Variable, Null oder nil sein. Wenn die in *KeyValues* übergebene Anzahl von Werten kleiner als die Anzahl der für die Suche verwendeten Indexspalten ist, wird für die fehlenden Werte Null verwendet.

Der Schlüssel muss immer ein Index sein, der in der Eigenschaft *IndexName* angegeben werden kann. Wenn für *IndexName* kein Wert angegeben wurde, verwendet *FindKey* den Id-Index der Tabelle.

Wenn die Suche erfolgreich war, positioniert *FindKey* den Cursor auf dem gefundenen Datensatz und gibt den Wert True zurück. Andernfalls wird der Cursor nicht verschoben, und *FindKey* gibt den Wert False zurück.

1.2.3.84 TTdbTable.FindNearest

Die Methode FindNearest bewegt den Cursor zum ersten Datensatz, der die größte Übereinstimmung mit den angegebenen Schlüsselwerten aufweist.

Delphi Syntax:

```
procedure FindNearest(const KeyValues: array of const);
```

C++ Syntax:

```
void __fastcall FindNearest(const System::TVarRec* KeyValues, const int
KeyValues_Size);
```

Beschreibung

Mit *FindNearest* kann der Cursor zu einem bestimmten Datensatz in einer Datenmenge oder zum ersten Datensatz der Datenmenge bewegt werden, der größer als die im Parameter *KeyValues* angegebenen Werte ist. *KeyValues* enthält ein Komma-Getrenntes-Array mit Feldwerten, das als Schlüssel bezeichnet wird. Bei den Werten des Schlüssels kann es sich um Literale, Variablen, Null, oder nil handeln. Wenn die in *KeyValues* übergebene Anzahl der Werte kleiner als die Anzahl der Spalten des für die Suche verwendeten Index ist, wird für die fehlenden Werte Null verwendet.

Der Schlüssel muss immer ein Index sein, der in der Eigenschaft *IndexName* angegeben werden kann. Wenn für *IndexName* kein Wert angegeben wurde, verwendet *FindKey* den Id-Index der Tabelle.

FindNearest positioniert den Cursor entweder auf einem Datensatz, der genau mit den Suchkriterien übereinstimmt oder auf dem ersten Datensatz, dessen Werte größer als die in den Suchkriterien angegebenen Werte sind.

1.2.3.85 TTdbTable.FlushMode

Schaltet den Schreibpuffer an und ab.

Delphi Syntax:

```
property FlushMode: TTdbFlushMode;
```

C++ Syntax:

```
__property TTdbFlushMode FlushMode = {read=FFlushMode, write=FFlushMode,
nodefault};
```

Beschreibung

Falls *FlushMode* auf *tfmSecure* gesetzt ist, werden die Datensätze auf die Festplatte geschrieben und die Verzeichnisinformation wird sofort aktualisiert. Dieser Modus minimiert den Datenverlust, bei einer plötzlich auftretenden Störung, wie einem Stromausfall oder einem Programmabsturz. Um die Performanz einer Anwendung zu verbessern, kann *FlushMode* auf *tfmFast* gesetzt und damit das Puffern von Schreibvorgängen aktiviert werden. *tfmDefault* verwendet die *FlushMode* Einstellung der Datenbank Komponente.

1.2.3.86 TTdbTable.ForeignKeyDefs

Enthält die Informationen zu den Fremdschlüsseln einer Tabelle.

Delphi Syntax:

```
property ForeignKeyDefs: TTdbForeignKeyDefs;
```

C++ Syntax:

```
__property TdbDataSet::TTdbForeignKeyDefs* ForeignKeyDefs =
{read=FForeignKeyDefs, write=SetForeignKeyDefs, stored=FStoreDefs};
```

Beschreibung

ForeignKeyDefs ist eine Kollektion von Fremdschlüssel Definitionen, wovon jede eine Fremdschlüssel Bedingung für die Tabelle beschreibt. Die Fremdschlüssel für eine Tabelle sind vor dem Aufruf von *CreateTable* oder *AlterTable* zu definieren.

Falls *ForeignKeyDefs* aktualisiert oder manuell editiert wird, wird das *StoreDefs* Property True.

Hinweis: Die Fremdschlüssel-Definitionen in *ForeignKeyDefs* müssen nicht immer die aktuellen Fremdschlüssel der Tabelle beschreiben. Daher muss immer die *Update* Method aufgerufen werden um die Liste zu aktualisieren.

1.2.3.87 TTdbTable.FulltextIndexDefs

Beinhaltet die Information über die Volltext-Indexe einer Tabelle.

Delphi Syntax:

```
property FulltextIndexDefs: TTdbFulltextIndexDefs;
```

C++ Syntax:

```
__property TdbDataSet::TTdbFulltextIndexDefs* FulltextIndexDefs =
{read=FFullltextIndexDefs, write=SetFulltextIndexDefs,
stored=FStoreDefs};
```

Beschreibung

FulltextIndexDefs ist eine Kollektion von Volltext-Index Definitionen, wobei jede einen verfügbaren Volltext-Index der Tabelle beschreibt. Die Definitionen sind vor dem Aufruf von *CreateTable* oder dem Erzeugen der Tabelle zur Entwurfszeit festzulegen.

Falls *FulltextIndexDefs* aktualisiert oder manuell editiert wird, wird die *StoreDefs* Property True.

Hinweis: Die Definitionen in *FulltextIndexDefs* müssen nicht immer die aktuellen Volltext-Index der Tabelle beschreiben. Vor einer Abfrage ist daher immer die Update-Methode der Kollektion aufzurufen.

1.2.3.88 TTdbTable.FullTextTable

Die TurboDB Tabelle, die als Index zur Volltextsuche verwendet wird.

Delphi Syntax:

```
property FullTextTable: TTdbTable;
```

C++ Syntax:

```
__property TTdbTable* FullTextTable = {read=FFullTextTable, write
=FFullTextTable};
```

Beschreibung

Setzen Sie *FullTextTable* auf die Tabelle, die die Schlüsselwörter zur Volltextsuche beinhaltet. Die Tabelle in der die Suche durchgeführt werden soll muss über eine Relationsfeld an *FullTextTable* gelinkt sein.

Um einen Volltextfilter anwenden zu können, muss WordFilter einen gültigen Volltextfilterausdruck beinhalten.

1.2.3.89 TTdbTable.GetIndexNames

Ermittelt eine Liste der verfügbaren Index einer Tabelle.

Delphi Syntax:

```
procedure GetIndexNames(List: TStrings);
```

C++ Syntax:

```
void __fastcall GetIndexNames(Classes::TStrings* List);
```

Beschreibung

Benutzen Sie *GetIndexNames* um eine Liste aller verfügbaren Indexe einer Tabelle zu ermitteln. List ist ein Stringlisten-Objekt, das von der Anwendung erzeugt und verwaltet wird und in das die Index-Namen eingetragen werden.

1.2.3.90 TTdbTable.GetUsage Method

Liefert Informationen über die aktuelle Benutzung der Tabelle durch alle beteiligten Anwendungen.

Delphi Syntax:

```
procedure GetUsage(out TableUsage: TTdbTableUsage);
```

C++ Syntax:

```
void __fastcall GetUsage(TTdbTableUsage& TableUsage);
```

Beschreibung

Die Funktion liefert Informationen über alle Anwendungen, die die Tabelle zum Zeitpunkt des Aufrufs geöffnet halten und den Status der Sperren.

Siehe auch

TTdbTableUsage Typ

1.2.3.91 TTdbTable.GotoKey

Die Methode *GotoKey* verschiebt den Cursor auf einen über den aktuellen Schlüssel angegebenen Datensatz.

Delphi Syntax:

```
function GotoKey: Boolean;
```

C++ Syntax:

```
bool __fastcall GotoKey(void);
```

Beschreibung

Mit GotoKey können Sie zu einem Datensatz wechseln, der mit Schlüsselwerten angegeben wird, die durch vorhergehende Aufrufe von SetKey oder EditKey und die Angabe der Suchwerte in der Eigenschaft Fields definiert wurden.

Wenn GotoKey einen übereinstimmenden Datensatz findet, wird der Cursor auf diesem positioniert und der Wert True zurückgegeben. Andernfalls wird die aktuelle Cursorposition nicht geändert, und GotoKey gibt den Wert False zurück.

1.2.3.92 TTdbTable.GotoNearest

Die Methode *GotoNearest* positioniert den Cursor auf dem Datensatz, der die größtmögliche Übereinstimmung mit dem aktuellen Schlüssel aufweist.

Delphi Syntax:

```
procedure GotoNearest;
```

C++ Syntax:

```
void __fastcall GotoNearest(void);
```

Beschreibung

Mit dem Aufruf von *GotoNearest* kann der Cursor auf dem Datensatz positioniert werden, der entweder genau mit dem durch die Schlüsselwerte bezeichneten Datensatz übereinstimmt oder dessen Werte die angegebenen Werte überschreiten.

Vor dem Aufruf von *GotoNearest* muß die Anwendung Schlüsselwerte festlegen. Dies erfolgt durch den Aufruf von *SetKey* oder *EditKey*, um der Datenmenge den Status *dsSetKey* zuzuweisen. Mit der Methode *FieldByName* wird anschließend der Schlüsselpuffer mit Werten gefüllt.

1.2.3.93 TTdbTable.IndexDefs

Die Eigenschaft IndexDefs enthält Informationen über die Indizes einer Tabelle.

Delphi Syntax:

```
property IndexDefs: TIndexDefs;
```

C++ Syntax:

```
__property Db::TIndexDefs* IndexDefs = {read=FIndexDefs, write
=SetIndexDefs, stored=IndexDefsStored};
```

Beschreibung

IndexDefs ist eine Kollektion von Indexdefinitionen, die jeweils einen verfügbaren Index der Tabelle beschreiben. Die Indexdefinitionen einer Tabelle müssen vor dem Aufruf von *CreateTable* oder, während des Entwurfs, vor dem Erzeugen einer Tabelle definiert sein.

Hinweis

Die in *IndexDefs* enthaltenen Definitionen berücksichtigen nicht immer die für die Tabelle verfügbaren Indizes. Rufen Sie vor der Überprüfung von *IndexDefs* die Methode *Update* auf, um die Liste zu aktualisieren.

1.2.3.94 TTdbTable.IndexName

Die Eigenschaft *IndexName* bezeichnet einen Sekundärindex für die Tabelle.

Delphi Syntax

```
property IndexName: String;
```

C++ Syntax

```
__property AnsiString IndexName = {read=FIndexName, write=SetIndexName};
```

Beschreibung

Bestimmen Sie mit *IndexName* welcher Index zur sortierten Anzeige der Tabelle verwendet werden soll. Verwenden Sie einen Eintrag aus *IndexFiles*. Setzen Sie *IndexName* auf einen Leerstring, um die Datensätze in Ihrer physikalischen Reihenfolge anzuzeigen.

1.2.3.95 TTdbTable.Key

Beinhaltet den nummerischen Code zur Verschlüsselung der Tabelle.

Diese Property ist in TurboDB 5 nicht mehr vorhanden. Wenn Sie eine Anwendung von TurboDB 4 oder früher upgraden, beachten Sie bitte die Hinweise zum Upgrade.

1.2.3.96 TTdbTable.LangDriver

Obsolet, nicht mehr verwenden. Siehe Kollationen und Collation.

Bezeichnet den Sprachtreiber, der für diese Tabelle verwendet wird.

Delphi Syntax:

```
property LangDriver: String;
```

C++ Syntax:

```
__property AnsiString LangDriver = {read=FLangDriver, write
=FLangDriver};
```

Beschreibung

Diese Eigenschaft liefert Ihnen die Sprache, an die die Datenbanktabelle gebunden ist. Der Wert der Eigenschaft ist identisch mit der Dateierweiterung der Sprachtreiber DLL die mit dieser Tabelle verwendet wird. Falls *LangDriver* beispielsweise *fra* enthält, verwendet die Tabelle die Dynamische

Link Bibliothek mit Namen *TdbLanDr.fra*.

Setzen Sie diese Eigenschaft um die Sprache zu bestimmen, die für die Tabelle in einem nachfolgenden Aufruf von *CreateTable* oder *AlterTable* festgelegt wird. Der Sprachtreiber muss zu diesem Zeitpunkt bereits existieren.

1.2.3.97 TTdbTable.LockTable

Sperrt eine TurboDB Tabelle.

Delphi Syntax:

```
procedure LockTable(LockType: TTdbLockType);
```

C++ Syntax:

```
void __fastcall LockTable(TTdbLockType LockType);
```

Beschreibung

Verwenden Sie *LockTable* damit eine Datenbanktabelle nicht durch eine andere Anwendung auf irgendeine Weise mit einer Sperre belegt werden kann. *LockType* spezifiziert den Typ der Sperre.

Das Einrichten einer Schreibsperre verhindert, dass andere Anwendungen schreibend auf die Tabelle zugreifen können. Eine Totalsperre verhindert jeden Zugriff durch eine andere Anwendung.

1.2.3.98 TTdbTable.MasterFields

Spezifiziert eines oder mehrere Felder in der Mastertabelle, die für die Verknüpfung mit den korrespondierenden Feldern der Detailtabelle herangezogen werden um eine Master-Detail Beziehung zwischen den Tabellen zu etablieren.

Delphi Syntax:

```
property MasterFields: String;
```

C++ Syntax:

```
__property AnsiString MasterFields = {read=GetMasterFields, write
=SetMasterFields};
```

Beschreibung

Verwenden Sie *MasterFields* nachdem die Eigenschaft *MasterSource* gesetzt wurde, um die Namen von einem oder mehreren Feldern zu definieren, die zur Detail-Master Verknüpfung zwischen dieser Tabelle und der über *MasterSource* spezifizierten verwendet werden soll.

MasterFields ist eine Zeichenkette, die einen oder mehrere Feldnamen der Mastertabelle enthält. Die Felder werden durch Semikolons voneinander getrennt. Immer wenn sich der aktuelle Datensatz der Mastertabelle ändert, werden die neuen Werte in diesen Feldern verwendet, um die korrespondierenden Datensätze in dieser Tabelle zur Anzeige zu selektieren.

Sie verwenden DetailFields, um die Felder der Datailtabelle zu definieren, die mit den Feldwerten der Mastertabelle übereinstimmen müssen.

Falls Sie die *MasterSource* Eigenschaft verwenden, *MasterFields* und *DetailFields* dagegen nicht, verwendet TurboDB die Standardbeziehung zwischen der Master- und Detailtabelle um die Verknüpfung zwischen den Tabellen zu etablieren. Die Standardbeziehung wird über Link- und Relationsfelder in den beiden Tabellen definiert.

1.2.3.99 TTdbTable.MasterSource

Referenziert die Master Datenquelle für eine Master-Detail Sicht.

Delphi Syntax:

```
property MasterSource: TDataSource;
```

C++ Syntax:

```
__property Db::TDataSource* MasterSource = {read=FMasterSource, write
=SetDataSource};
```

Beschreibung

Verwenden Sie *MasterSource* um den Namen der Datenquelle zu spezifizieren, deren *DataSet* Eigenschaft auf eine Datenmenge zeigt, die zur Etablierung einer Master-Detail Beziehung zwischen dieser und einer anderen Tabelle herangezogen werden soll. Immer wenn sich der aktuelle Datensatz in der Mastertabelle ändert, werden die verknüpften Datensätze in der Detailtabelle selektiert.

TurboDB kann die Verknüpfung zwischen einer Master- und einer Detailtabelle auf drei verschiedene Arten herstellen:

- Die Standardbeziehung wird mit Hilfe von Link- und Relationsfeldern definiert. Lassen Sie die Eigenschaften *MasterField* und *DetailFields* leer, um diese Art der Verknüpfung zu verwenden.
- Falls die korrespondierenden Felder in der Master- und Detailtabelle identische Namen haben, können Sie die Verknüpfung auch durch alleiniges setzen der *MasterFields* Eigenschaft etablieren.
- Um die Verknüpfung über Felder mit unterschiedlichen Namen herzustellen, können Sie zusätzlich zu *MasterFields* die Eigenschaft *DetailFields* verwenden.

Hinweis: Wählen Sie zur Entwurfszeit eine verfügbare Datenquelle aus der Auswahlliste in der *MasterSource* Eigenschaft im Objektinspektor aus.

Achtung: Alle Tabellen, für die eine Master-Detail Beziehung hergestellt werden soll müssen der selben Datenbank angehören

1.2.3.100 TTdbTable.Password

Spezifiziert das Passwort, das zum Öffnen der Tabelle verwendet werden soll.

Delphi Syntax:

```
property Password: String;
```

C++ Syntax:

```
__property AnsiString Password = {read=FPassword, write=FPassword};
```

Beschreibung

Setzen Sie *Password* vor dem Öffnen einer geschützten Tabelle. Falls die Tabelle zusätzlich verschlüsselt ist, müssen Sie auch noch *Key* setzen. Falls *Password* oder *Key* einen falschen Wert aufweisen, löst die Datenbankkomponente ein *OnPassword*-Ereignis aus. Falls keine Ereignisbehandlungsroutine definiert ist, kann die Tabelle nicht geöffnet werden und eine *ETurboDBError* Exception wird ausgelöst.

Wenn Sie eine Tabelle erzeugen oder ändern, werden die aktuellen Werte von EncryptionMethod und *Password* zum Schutz der Tabelle verwendet.

Falls die Verschlüsselungsmethode der Tabelle *temClassic* ist, wird sowohl ein alphanummerisches Passwort als auch ein 32-Bit-Code benötigt. In diesem Falls setzen Sie die Eigenschaft Password auf das alphanummerische Passwort gefolgt von einem Strichpunkt und der Zahl für den Code, zum Beispiel *MyPass;-7896879.* Es wird jedoch empfohlen, solche Tabelle auf die Verschlüsselungsmethode *FastEncrypt* umzustellen, um die Handhabung zu vereinfachen.

Siehe auch

EncryptionMethod

1.2.3.101 TTdbTable.ReadOnly

Die Eigenschaft ReadOnly gibt an, ob eine Tabelle in dieser Anwendung bearbeitet werden kann.

Delphi Syntax:

```
property ReadOnly: Boolean;
```

C++ Syntax:

```
__property bool ReadOnly = {read=FReadOnly, write=SetReadOnly,
nodefault};
```

Beschreibung

Setzen Sie *ReadOnly* auf True, falls Sie die Datenquelle im Readonly Modus öffnen möchten. Wird daraufhin eine Methode aufgerufen, die Daten ändern möchte, wird eine Exception ausgelöst.

1.2.3.102 TTdbTable.RenameTable

Die Methode RenameTable benennt die TurboDB-Tabelle um, die mit dieser Tabellenkomponente verknüpft ist.

Delphi Syntax:

```
procedure RenameTable(const NewTableName: String);
```

C++ Syntax:

```
void __fastcall RenameTable(const AnsiString NewTableName);
```

Beschreibung

Durch den Aufruf von RenameTable kann der zugrundeliegenden TurboDB-Tabelle ein neuer Name zugewiesen werden. RenameTable benennt die Tabelle und die zugehörigen Dateien um. Dazu gehören die Standard Index Dateien sowie die Memo- und Blobdateien. Verwenden Sie diese Methode mit Vorsicht, da alle Anwendungen die diese Datenbanktabelle verwenden betroffen sind.

1.2.3.103 TTdbTable.RepairTable

Repariert eine Datenbank-Tabelle.

Delphi Syntax:

```
procedure RepairTable;
```

C++ Syntax:

```
void __fastcall RepairTable();
```

Beschreibung

Setzen Sie *RepairTable* ein, wenn die Tabelle Fehler wie inkorrekt Verbindungen zu Memos und Blobs enthält. Die Tabelle wird neu aufgebaut und dadurch eventuell vorhandene Fehler soweit möglich korrigiert. Machen Sie eine Sicherungskopie der Tabelle, bevor Sie diese Funktion aufrufen.

1.2.3.104 TTdbTable.SetNextAutoIncValue

Bestimmt den Startwert für die nachfolgende Vergabe von *AutoInc* Werten.

Delphi Syntax:

```
procedure SetNextAutoIncValue(FieldNo: SmallInt; Value: LongInt);
```

C++ Syntax:

```
void __fastcall SetNextAutoIncValue(SmallInt FieldNo, LongInt Value);
```

Beschreibung

Verwenden Sie diese Funktion um den Startwert für die weitere Vergabe von Werten für ein AutoInc Feld festzulegen. Verwenden Sie diese Methode mit Bedacht. Ein Wert der kleiner ist als die bereits in der Tabelle vorhandenen, kann zu doppelten AutoInc Werten führen. Besonders hilfreich ist der Einsatz dieser Funktion nach einem Import via *BatchMove* mit *RecalcAutoInc* gleich False.

1.2.3.105 TTdbTable.SetKey

Mit der Methode SetKey können Schlüssel und Bereiche für eine Datenmenge vor dem Beginn einer Suche gesetzt werden.

Delphi Syntax:

```
procedure SetKey;
```

C++ Syntax:

```
void __fastcall SetKey(void);
```

Beschreibung

Mit Hilfe von SetKey kann der Datenmenge der Status dsSetKey zugewiesen und der aktuelle Inhalt des Schlüsselpuffers gelöscht werden. Über die Methode FieldByName können dann neue Suchwerte festgelegt werden.

Hinweis

Mit dem Aufruf von EditKey kann ein vorhandener Schlüssel oder Bereich geändert werden.

1.2.3.106 TTdbTable.TableFileName

Bezeichnet den vollständigen Dateipfad der zugrunde liegenden Datenbanktabelle.

Delphi Syntax:

```
property TableFileName: String;
```

C++ Syntax:

```
__property AnsiString TableFileName = {read=GetTableFileName};
```

Beschreibung

TableFileName ist eine Eigenschaft die den Dateinamen der Datenbanktabelle inklusive Laufwerk (Windows) und Pfad liefert. Die Eigenschaft kann nur gelesen werden.

1.2.3.107 TTdbTable.TableLevel

Bezeichnet die Version der Datenbanktabelle.

Delphi Syntax:

```
property TableLevel: Integer;
```

C++ Syntax:

```
__property int TableLevel = {read=FTableLevel, write=SetTableLevel,
nodefault};
```

Beschreibung

Lesen Sie *TableLevel* um die Version der Datenbanktabelle zu bestimmen. Einige Fähigkeiten der Tabelle hängen vom *TableLevel* ab. Zum Beispiel sind *DateTime* Felder erst ab *TableLevel* 3 verfügbar. Vor dem Erzeugen oder Ändern der Tabellenstruktur müssen Sie *TableLevel* auf den gewünschten Wert setzen. Eine Beschreibung der verschiedenen Level finden Sie im Kapitel " Tabellen Level".

1.2.3.108 TTdbTable.TableName

Die Eigenschaft *TableName* gibt den Namen der Datenbanktabelle an, die diese Komponente kapselt.

Delphi Syntax:

```
property TableName: TFileName;
```

C++ Syntax:

```
__property AnsiString TableName = {read=FTableName, write=SetTableName};
```

Beschreibung

In *TableName* geben Sie den Namen der Datenbanktabelle an, die diese Komponente kapselt. Bevor Sie *TableName* einen gültigen Wert zuweisen, sollte die Eigenschaft *DatabaseName* bereits gesetzt sein. Wenn *DatabaseName* während des Entwurfs gesetzt wird, kann ein gültiger Tabellenname in der TableName-Dropdown-Liste im Objektinspektor gewählt werden.

Hinweis

Damit *TableName* ein Wert zugewiesen werden kann, muss die Eigenschaft *Active* den Wert False haben.

1.2.3.109 TTdbTable.UnlockTable

Die Methode *UnlockTable* entfernt eine vorher zugewiesene Sperre von einer Tabelle.

Delphi Syntax:

```
procedure UnlockTable(LockType: TTdbLockType);
```

C++ Syntax:

```
void __fastcall UnlockTable(TTdbLockType LockType);
```

Beschreibung

Mit dem Aufruf von *UnlockTable* können Sie die einer Tabelle zugewiesene Sperre aufheben. *LockType* gibt die zu entfernende Sperre an.

Das Entfernen einer Schreibsperre ermöglicht den Schreibzugriff auf die Tabelle durch andere Anwendungen. Das Entfernen einer Totalsperre ermöglicht anderen Anwendungen sowohl Schreib- als auch Lesezugriff auf die Tabelle.

1.2.3.110 TTdbTable.UpdateFullTextIndex

Erstellt einen bestehenden Volltextindex neu.

Delphi Syntax:

```
procedure UpdateFulltextIndex(const Name: AnsiString); overload;
```

```
procedure UpdateFullTextIndex(const Fields, RelationField,
CounterIndexFileName: String; Limit: Integer); overload;
```

C++ Syntax:

```
void __fastcall UpdateFulltextIndex(const AnsiString Name);
```

```
void __fastcall UpdateFullTextIndex(const AnsiString Fields, const
AnsiString RelationField, const AnsiString CounterIndexFileName, int
Limit);
```

Beschreibung

Verwenden Sie *UpdateFullTextIndex* um Ihre Tabelle nach Schlüsselwörtern zu durchsuchen und sie in die Schlüsselworttabelle aufzunehmen. Nach der Aktion können sie die Schlüsselworttabelle zur Volltextsuche und als Filter verwenden. Dier erste Version der Methode kann nur für neue Volltext-Indexe (Tabellen-Level 4 und höher) benutzt werden um diese zu reparieren. Die zweite Version gilt für alte (ungewartete) Volltext-Indexe (Tabellen-Level bis 3). Durch den Aufruf dieser Funktion werden dem Volltext-Index neue Datensätze hinzugefügt und nicht mehr vorhandene

entfernt.

Fields ist eine durch Komma getrennte Liste derjenigen Felder, die in den Index aufgenommen werden.

RelationField ist der Name des Relations-Feldes, das die Verbindung zur Volltext-Tabelle herstellt.

CounterIndexFileName enthält einen optionalen Dateinamen. Diese Datei enthält pro Zeile ein Wort, das aus dem Volltext-Index ausgeschlossen werden soll.

Limit enthält eine Obergrenze für die Fundstellen eines einzelnen Schlüsselwortes. Überschreitet ein Wort diese Obergrenze, wird es nicht in den Volltext-Index aufgenommen.

1.2.3.111 TTdbTable.UpdateIndex

Erstellt einen bestehenden Index neu.

Delphi Syntax:

```
procedure UpdateIndex(const Name: String);
```

C++ Syntax:

```
void __fastcall UpdateIndex(const AnsiString Name);
```

Beschreibung

Verwenden Sie *UpdateIndex* um den Index einer Tabelle neu zu erzeugen. Das kann nötig werden, wenn der Index zu fehlerhaften Such- und Sortierergebnissen führt.

1.2.3.112 TTdbTable.WordFilter

Definiert den Ausdruck für die Volltextsuche

Delphi Syntax:

```
property WordFilter: WideString;
```

C++ Syntax:

```
__property WideString WordFilter = {read=FWordFilter, write
=SetWordFilter};
```

Beschreibung

Verwenden Sie *WordFilter*, wenn Sie eine Tabelle nach einem oder mehreren Schlüsselwörtern filtern möchten.

Es muss ein Volltextindex für die Tabelle existieren und *FullTextTable* muss auf die Tabellenkomponente gesetzt sein, die die Schlüsselwörter enthält. Falls *WordFilter* und *Filter* (oder *FilterW*) verwendet werden, werden nur die Datensätze angezeigt, die beiden Filtern genügen. Mehr über die Volltext-Indizierung erfahren Sie in "Einen Volltextindex zur Laufzeit benutzen" und in "Volltext Suchbedingungen".

1.2.3.113 TTdbTableFormat

Gibt das Dateiformat einer Tabellendatei an.

Unit

TdbTypes

Delphi Syntax:

```
type TTdbTableFormat = (tffDBase, tffSDF, tffMyBase, tffTdb);
```

C++ Syntax:

```
enum TTdbTableFormat {tffDBase, tffSDF, tffMyBase, tffTdb};
```

Beschreibung

Dateiform Beschreibung

at

tffDBase	dBase III+ kompatible Datei
tffSdf	System Data Format. Eine Textdatei, ein Trennzeichen trennt die Werte, die durch Anführungszeichen eingeschlossen sind.
tffMyBase	XML Datei im MyBase Format. Das ist das Format das *TClientDataSet* benutzt um Daten zu speichern. dieses Format wird nur zum Export unterstützt.
tffTdb	TurboDB Datenbank Tabellendatei

1.2.3.114 TTdbTableUsage Type

Beschreibung des aktuellen Zugriffs auf die Tabelle.

Delphi Syntax:

```
TTdbTableUsage = record
  ItemCount: Integer;
  UserCount: Integer;
  ReadCount: Integer;
  UpgradeCount: Integer;
  WriteCount: Integer;
  WaitCount: Integer;
  WaitWriteCount: Integer;
  UpdateCount: Integer;
  UserInfo: array[0..MaxTableUsers-1] of TTdbUsageUserInfo;
end;
```

C++ Syntax:

```
class TTdbTableUsage {
  int ItemCount;
  int UserCount;
  int ReadCount;
  int UpgradeCount;
  int WriteCount;
  int WaitCount;
  int WaitWriteCount;
  int UpdateCount;
  TTdbUsageUserInfo UserInfo[MaxTableUsers];
};
```

Beschreibung

ItemCount	Anzahl der für die Tabelle registrierten Anwendungen
UserCount	Anzahl der aktiven Sessions
ReadCount	Anzahl der Sessions mit lesendem Zugriff
UpgradeCount	Anzahl der Sessions mit lesendem Zugriff und zu erwartenden Schreibzugriff
WriteCount	Anzahl der Sessions mit schreibenden Zugriff
WaitCount	Anzahl der Sessions die auf lesenden Zugriff warten
WaitWriteCount	Anzahl der Sessions die auf schreibenden Zugriff warten
UpdateCount	Anzahl der durchgeführten Änderungen seitdem die Tabelle durch die erste Anwendung geöffnet wurde
UserInfo	Informationen zu jeder Session

1.2.3.115 TTdbUsageUserInfo

Beschreibung des aktuellen Zugriffs auf eine Datenbank-Tabelle.

Delphi Syntax:

```
TTdbUsageUserInfo = record
  ConnectionName: ShortString;
  ConnectionId: LongWord;
  Active: Boolean;
  LockCount: Integer;
  TimeOut: Integer;
  RecordNo: Integer;
end;
```

C++ Syntax:

```
class TTdbUsageUserInfo {
  ShortString ConnectionName;
  unsigned int ConnectionId;
  bool Active;
  int LockCount;
  int TimeOut;
  int RecordNo;
};
```

Beschreibung

ConnectionName	Der benutzerdefinierte Name der Session/Connection
ConnectionId	Vom System vergebene eindeutige Id für die Session/Connection
Active	Session ist aktiv
LockCount	Anzhal der Lese-Sperren
TimeOut	Time-out
RecordNo	Physikalische Nummer des gesperrten Datensatzes

1.2.3.116 TTdbEncryptionMethod

Beschreibt die Art wie eine Tabelle geschützt oder verschlüsselt ist.

unit

TdbTypes

Delphi Syntax:

```
type TTdbEncryptionMethod = (temDefault, temNone, temProtection,
temClassic, temFastEncrypt, temBlowfish, temRijndael);
```

C++ Syntax:

```
enum TTdbEncryptionMethod {temDefault, temNone, temProtection,
temClassic, temFastEncrypt, temBlowfish, temRijndael};
```

Beschreibung

Die Werte dieses Typs dienen zur Beschreibung auf welche Art eine TurboDB Datenbank oder Tabelle gegen nicht autorisierten Zugriff geschützt ist:

Wert	Beschreibung
temDefault	Nur für Tabellen erlaubt, nicht für Datenbanken. Die Tabelle verwendet die allgemeinen Einstellungen der Datenbank für die Verschlüsselung.
temNone	Die Tabelle ist nicht geschützt.
temProtection	Es gibt ein Passwort für die Tabelle, aber die Tabelle ist nicht verschlüsselt.
temClassic	Es gibt ein Passwort plus einen 32 Bit Schlüssel.
temFastEncrypt	Schnelle Verschlüsselung die einen von einem Passwort abgeleiteten 32 Bit Schlüssel verwendet.

temBlowfish	Blowfish Verschlüsselung mit einem von einem Passwort abgeleiteten 128 Bit Schlüssel.
temRijndael	Rijndael (AES) Verschlüsselung mit einem von einem Passwort abgeleiteten 128 Bit Schlüssel.

Siehe auch

Datensicherheit

1.2.3.117 TTdbBatchMove

Überträgt Datensätze zwischen TurboDB Tabellen und anderen Datenquellen wie zum Beispiel dBase Tabellen oder beliebigen Nachkommen von *TDataSet*.

Unit

TdbBatchMove

Beschreibung

TTdbBatchMove importiert Datensätze von anderen Datasets oder von Dateien und exportiert Datensätze in verschiedene Dateitypen. Die Dateitypen sind:

- Text/SDF
- TurboDB
- dBase
- XML/MyBase (nur Export)

Außerdem können beliebige *TDataSet* Nachkommen als externe Datenquelle verwendet werden.

1.2.3.118 TTdbBatchMove Hierarchy

Hierarchie

TComponent

|

TTdbBatchMove

1.2.3.119 TTdbBatchMove Methods

In TTdbBatchMove

Execute

Abgeleitet von TComponent

(In der Embarcadero Dokumentation finden Sie weitere Informationen)

1.2.3.120 TTdbBatchMove Properties

In TTdbBatchMove

CharSet

ColumnNames

DataSet

Direction

FileName

FileType
Filter
Mappings
Mode
MovedCount
ProblemCount
Quote
RecalcAutoInc
Separator
TdbDataSet

Abgeleitet von TComponent

(In der Embarcadero Dokumentation finden Sie weitere Informationen)

1.2.3.121 TTdbBatchMove Events

In TTdbBatchMove

OnProgress

Abgeleitet von TComponent

(In der Embarcadero Dokumentation finden Sie weitere Informationen)

1.2.3.122 TTdbBatchMove.CharSet

Definiert den Zeichensatz für Textdateien und dBase Tabellen.

Delphi Syntax:

```
TTdbCharSet = (tcsAnsi, tcsOem, tcsUtf8);
```

```
property CharSet: TTdbCharSet;
```

C++ Syntax:

```
enum TTdbCharSet {tcsAnsi, tcsOem, tcsUtf8};
```

```
__property Tdbtypes::TTdbCharSet CharSet = {read=FCharSet, write
=FCharSet, nodefault};
```

Beschreibung

Setzen Sie *CharSet* wenn Sie in eine Textdatei oder in eine dBase Tabelle exportieren möchten, um den Zeichensatz für die Exportdatei festzulegen. Beim Import ist *CharSet* wichtig, um Zeichenketten zu konvertieren, die aus der Quelldatei gelesen werden.

1.2.3.123 TTdbBatchMove.ColumnNames

Bestimmt ob eine Textdatei in der ersten Zeile die Spaltennamen beinhaltet.

Delphi Syntax:

```
property ColumnNames: Boolean;
```

C++ Syntax:

```
__property bool ColumnNames = {read=FColumnNames, write=FColumnNames,
nodefault};
```

Beschreibung

Die Eigenschaft wird nur berücksichtigt, wenn der BatchMove ein Import oder ein Export aus oder in eine Textdatei ist. Falls *Direction bmdExport* ist, teilt die Eigenschaft der TTdbBatchMove Komponente mit, die Spaltennamen in die erste Zeile der Exportdatei zu schreiben. Falls *Direction* auf *bmdImport* gesetzt ist und *ColumnNames* true ist, wird die erste Zeile der Importdatei als Spaltennamen interpretiert.

1.2.3.124 TTdbBatchMove.DataSet

Bestimmt ein Dataset als Quelle für den BatchMove.

Delphi Syntax:

```
property DataSet: TDataSet;
```

C++ Syntax:

```
__property Db::TDataSet* DataSet = {read=FDataSet, write=FDataSet};
```

Beschreibung

Verwenden Sie die Eigenschaft, wenn nicht eine Datei, sondern ein DataSet die Quelle für den BatchMove ist. Falls die Quelle eine Datei ist, verwenden Sie stattdessen die Eigenschaft FileName.

1.2.3.125 TTdbBatchMove.Direction

Bestimmt die Richtung des Datentransfers.

Delphi Syntax:

```
TBatchMoveDirection = (bmdImport, bmdExport);
```

```
property Direction: TBatchMoveDirection
```

C++ Syntax:

```
enum TBatchMoveDirection {bmdImport, bmdExport};
```

```
__property TBatchMoveDirection Direction = {read=FDirection, write
=FDirection, nodefault};
```

Beschreibung

Setzen Sie die Eigenschaft um zwischen Import und Export von Datensätzen zu wechseln. Ein Import ist ein Datentransfer zu einer TdbTable, Export bedeutet Daten aus einer TdbTable zu transferieren. Ein Export kann nur in eine Datei erfolgen, nicht in ein Dataset.

1.2.3.126 TTdbBatchMove.Execute

Mit der Methode *Execute* können Sie die Batch-Operation durchführen, die durch Mode und Directionfestgelegt ist.

Delphi Syntax:

```
procedure Execute;
```

C++ Syntax:

```
void __fastcall Execute(void);
```

Beschreibung

Nachdem Sie in den Eigenschaften angegeben haben, welche Batch-Operation wie durchgeführt werden soll, können Sie die Operation mit *Execute* starten. Sie müssen mindestens die Eigenschaften FileName, FileType, Mode und Direction einstellen.

1.2.3.127 TTdbBatchMove.FileName

Bestimmt den Dateinamen einer anderen Datenquelle.

Delphi Syntax:

```
property FileName: String;
```

C++ Syntax:

```
__property AnsiString FileName = {read=FFileName, write=FFileName};
```

Beschreibung

Verwenden Sie die *FileName* um den Dateinamen der externen Datei für den Import oder Export festzulegen.

Falls Direction *bmdImport* ist, muss die Datei existieren und Datensätze in dem Format beinhalten, das durch die Eigenschaft FileType eingestellt ist. Falls *Direction bmdExport* ist, wird die Datei durch Aufruf der Execute Methode erzeugt. Eine existierende Datei wird in diesem Fall überschrieben.

1.2.3.128 TTdbBatchMove.FileType

Bestimmt das Dateiformat der Ziel/Quelldatei beim Datentransfer.

Delphi Syntax:

```
property FileType: TTdbTableFormat;
```

C++ Syntax:

```
__property Tdbtypes::TTdbTableFormat FileType = {read=FTableType, write
=FTableType, nodefault};
```

Beschreibung

Verwenden Sie diese Eigenschaft um das Dateiformat der Datenquelle anzugeben, aus der Datensätze importiert werden sollen oder geben Sie für einen Exports das zu erzeugende Dateiformat an.

Einige Formate können nur für Import oder Export verwendet werden. Falls der Dateityp für die gewünschte Aktion nicht unterstützt wird, wird Execute eine Exception auslösen.

1.2.3.129 TTdbBatchMove.Filter

Definiert eine Filterbedingung für die zu ex- oder importierenden Datensätze.

Delphi Syntax:

```
property Filter: String;
```

C++ Syntax:

```
__property AnsiString Filter = {read=GetFilter, write=SetFilter};
```

Beschreibung

Setzen Sie die Eigenschaft *Filter* um die zu übertragenden Datensätze einzuschränken. Filter ist eine Suchbedingung, wie sie in "Suchbedingungen" beschrieben ist.

1.2.3.130 TTdbBatchMove.Mappings

Die Eigenschaft legt die Spaltenzuordnungen für eine Batch-Import fest.

Delphi Syntax:

```
property Mappings: TStrings;
```

C++ Syntax:

```
__property Classes::TStrings* Mappings = {read=FMappings, write
=SetMappings};
```

Beschreibung

Durch Einstellen von Mappings können Sie die Zuordnungen zwischen den Feldern der Quelltabelle und den Feldern der Zieltabelle festlegen. Per Voreinstellung wird eine Zuordnung vorausgesetzt, die sich aus der Position der Felder in der Quell- und Zieltabelle ergibt. Die erste Spalte der Quelle wird also der ersten Spalte der Zieltabelle zugeordnet u.s.w. Mit Mappings kann in einer Anwendung diese Vorgabe umgangen werden.

Um die Spalte SourceColName der Quelltabelle der Spalte DestColName der Zieltabelle zuzuordnen, verwenden Sie folgende Syntax:

```
DestColName=SourceColName
```

Sie können auch die Spaltennummern verwenden:

```
$8=$3
```

Sie können die verschiedenen Arten der Spaltenbezeichnung auch mischen:

```
$8=SourceColName
```

Beim Einfügen oder Anhängen von Datensätzen werden die Felder in der Zieltabelle auf NULL gesetzt, die in Mappings nicht eingetragen sind. In der Kopie einer Datenmenge erscheinen dann solche Felder auch nicht als Spalten.

1.2.3.131 TTdbBatchMove.Mode

Die Eigenschaft Mode gibt das Verhalten des Objekts TBatchMode nach dem Aufruf der Methode Execute an.

Delphi Syntax:

```
property Mode: TTdbBatchMode;
```

C++ Syntax:

```
__property Tdbtypes::TTdbBatchMode Mode = {read=FMode, write=FMode,
nodefault};
```

Beschreibung

Mit der Eigenschaft Mode können Sie festlegen, ob das Objekt TBatchMove Datensätze einfügen, ersetzen oder löschen oder die Quelltabelle kopieren soll. Im folgenden finden Sie die möglichen Werte für Mode:

Wert	Bedeutung
tbmAppend	Die Datensätze der Quelltabelle werden in die Zieltabelle eingefügt. Die Zieltabelle muß bereits existieren, und die beiden Tabellen dürfen keine Datensätze mit doppelten Schlüsseln enthalten. Dies ist der Standardmodus.
tbmUpdate	Die Datensätze der Zieltabelle werden durch die übereinstimmenden Datensätze der Quelltabelle ersetzt. Die Zieltabelle muß existieren und mit einem geeigneten Index versehen sein.
tbmAppendUpdate	Falls in der Zieltabelle ein übereinstimmender Datensatz enthalten ist, wird er ersetzt. Andernfalls werden die Datensätze in die Zieltabelle eingefügt. Die Zieltabelle muß existieren und mit einem geeigneten Index versehen sein.
tbmCopy	Die Zieltabelle wird mit der Struktur der Quelltabelle erstellt. Falls die Zieltabelle bereits existiert, wird sie bei der Operation gelöscht und durch die neue Kopie der Quelltabelle ersetzt.
tbmDelete	Es werden die Datensätze der Zieltabelle gelöscht, die mit Datensätzen der Quelltabelle übereinstimmen. Die Zieltabelle muß bereits existieren und mit einem Index versehen sein.

Hinweis: *TTdbBatchMode* ist in Unit *TdbTypes* definiert.

1.2.3.132 TTdbBatchMove.MovedCount

Die Eigenschaft *MovedCount* gibt die Anzahl der Datensätze der Quelltabelle an, die in die Zieltabelle eingetragen wurden.

Delphi Syntax:

```
property MovedCount: LongInt;
```

C++ Syntax:

```
__property int MovedCount = {read=FMovedCount, nodefault};
```

Beschreibung

Mit der Eigenschaft *MovedCount* können Sie die Anzahl der Datensätze der Quelltabelle ermitteln, die während der Ausführung der Methode *Execute* gelesen wurden. Der Wert von *MovedCount* enthält keine Datensätze, die durch eine Filterbedingung ausgeschlossen wurden.

1.2.3.133 TTdbBatchMove.OnProgress

Das Ereignis OnProgress wird während der Operationen in regelmäßigen Abständen ausgelöst.

Delphi Syntax:

```
TTdbProgressEvent = procedure(Sender: TObject; PercentDone: Byte; var
Stop: Boolean) of object;

property OnProgress: TBatchMoveProgress;
```

C++ Syntax:

```
__property Tdbdataset::TTdbProgressEvent OnProgress = {read=FOnProgress,
write=FOnProgress};
```

Beschreibung

Schreiben Sie eine Ereignisbehandlungsroutine für *OnProgress* um den Fortschritt der Aktion auf dem Bildschirm darzustellen.

PercentDone beinhaltet die Prozentangabe der bereits bearbeiteten Datensätze. Durch setzen des *Stop* Argumentes auf True, können Sie den BatchMove abbrechen.

Hinweis

Führen Sie keine TurboDB Methoden in der Ereignisbehandlungsroutine aus, da es sonst zu Störungen des Datentransfers kommen kann.

1.2.3.134 TTdbBatchMove.ProblemCount

Die Eigenschaft ProblemCount gibt die Anzahl der Datensätze an, die wegen fehlender Übereinstimmung des Feldtyps nicht ohne Datenverlust in die Zieltabelle eingefügt werden konnten.

Delphi Syntax:

```
property ProblemCount: LongInt;
```

C++ Syntax:

```
__property int ProblemCount = {read=FProblemCount, nodefault};
```

Beschreibung

Mit ProblemCount können Sie die Anzahl der Datensätze der Quelltabelle mit Feldwerten ermitteln, die keinem Feld der Zieltabelle zugeordnet werden konnten, ohne daß sie abgeschnitten wurden.

Falls die Datenquelle eine Textdatei ist geschieht dies zum Beispiel, wenn die Quelldaten nicht mit dem Datentyp einer Zielspalte übereinstimmt. So kann das Zeichen 'a' nicht in eine Datumspalte geschrieben werden.

1.2.3.135 TTdbBatchMove.Quote

Bestimmt das Zeichen, das in einer Textdatei verwendet wird um Strings zu umschließen.

Delphi Syntax:

```
property Quote: Char;
```

C++ Syntax:

```
__property char Quote = {read=FQuote, write=FQuote, nodefault};
```

Beschreibung

Setzen Sie *Quote* auf einen Wert ungleich #0 um Zeichenketten in der Exportdatei zu umschließen. Beim Import ist *Quote* wichtig, um den korrekten Wert von Zeichenketten der Importdatei zu ermitteln. *Quote* wird nur berücksichtigt, falls *FileType* auf *tffSDF* gesetzt ist.

1.2.3.136 TTdbBatchMove.RecalcAutoInc

Definiert, ob Auto-Nummern beim Import neu berechnet werden sollen.

Delphi Syntax:

```
property RecalcAutoInc: Boolean;
```

C++ Syntax:

```
__property bool RecalcAutoInc = {read=FRecalcAutoInc, write=
FRecalcAutoInc, default=0}
```

Beschreibung

Setzen Sie diese Eigenschaft auf True, wenn Sie Datensätze mit Auto-Nummern importieren möchten und wenn diese Datensätze Werte haben, die in der Datenbank schon vorhanden sind. Falls die Eigenschaft auf True gesetzt ist, berechnet TurboDB neue eindeutige Werte für die importieren AutoInc-Felder. Falls die Eigenschaft False ist, versucht TurboDB die Auto-Nummern aus der Quelltabelle zu beizubehalten. In diesem Fall kann es passieren, dass Ihre Tabelle nach dem Import doppelte Auto-Nummern enthält. Verwenden Sie die Methode *SetNextAutoIncValue* um festzulegen, mit welchem Wert die AutoInc-Werte nach dem Import weitergeführt werden sollen.

Hinweis

Falls Ihre Anwendung Auto-Nummern verwendet um Beziehungen zwischen Tabellen herzustellen, werden Sie die Verknüpfungen zwischen den Tabellen verlieren, wenn Sie *RecalcAutoInc* auf True setzen.

1.2.3.137 TTdbBatchMove.Separator

Definiert das Zeichen durch das die Felder in einer Textdatei getrennt sind.

Delphi Syntax:

```
property Separator: Char;
```

C++ Syntax:

```
__property char Separator = {read=FSeparator, write=FSeparator,
nodefault};
```

Beschreibung

Setzen Sie *Separator* um die Feldwerte beim Import einer Textdatei richtig zu analysieren. Beim Export wird das Zeichen als Trennzeichen zwischen den einzelnen Spalten verwendet. *Separator* wird nur im Zusammenhang mit Textdateien berücksichtigt, wenn FileType also auf *tffSDF* gesetzt ist.

1.2.3.138 TTdbBatchMove.TdbDataSet

Bestimmt die TurboDB DataSet-Komponente, die als Quelle oder als Ziel eines Datentransfers verwendet wird.

Delphi Syntax:

```
property TdbDataSet: TTdbDataSet;
```

C++ Syntax:

```
__property Tdbdataset::TTdbDataSet* TdbDataSet = {read=FTdbDataSet,
write=FTdbDataSet};
```

Beschreibung

Setzen Sie *TdbDataSet* auf die Tabellenkomponente in die Sie Datensätze importieren oder aus der Sie Datensätze exportieren möchten. Falls Direction gleich *bmdImport* ist, werden Datensätze aus der externen Datenquelle FileName oder DataSet in die durch *TdbTable* definierte Tabelle übertragen. Falls Direction gleich *bmdExport* ist, werden Datensätze aus der Tabellenkomponente in die Datei geschrieben.

1.2.3.139 TTdbDatabase

TTdbDatabase kontrolliert eine Verbindung zu einer Datenbank in einer TurboDB basierten Anwendung.

Unit

TdbDataSet

Beschreibung

Verwenden Sie *TTdbDatabase* falls Ihre TurboDB basierte Datenbankanwendung eine der folgenden Merkmale benötigt:

- Allgemeines Datenbank Verzeichnis für zwei oder mehr TurboDB Tabellen
- Passwortabfrage um geschützte Tabellen zu öffnen
- Wiederherstellung früher benutzter Verbindungen

Anmerkung

Die explizite Deklaration einer *TTdbDatabase* Komponente für jede Datenbankverbindung in einer Anwendung ist optional falls die Anwendung nicht explizit die Kontrolle über die Verbindung benötigt. Falls keine *TTdbDatabase* Komponente für eine Datenbankverbindung deklariert und instantiiert wurde, wird zur Laufzeit eine temporäre Datenbank Komponente mit Standardeigenschaften erzeugt.

1.2.3.140 TTdbDatabase Hierarchy

Hierarchie

TObject
|
TPersistent
|
TComponent
|
TCustomConnection
|
TTdbDatabase

1.2.3.141 TTdbDatabase Methods

In TTdbDatabase

Backup
CloseCachedTables
CloseDataSets
Commit
Compress
RefreshDataSets
Rollback
StartTransaction

Abgeleitet von TCustomConnection

(In der Embarcadero Dokumentation finden Sie weitere Informationen)

1.2.3.142 TTdbDatabase Properties

In TTdbDatabase

BlobBlockSize
CacheSize
ConectionId
ConnectionName
DatabaseName
Exclusive
FlushMode
IndexPageSize
Location
PrivateDir

Abgeleitet von TCustomConnection

(In der Embarcadero Dokumentation finden Sie weitere Informationen)

1.2.3.143 TTdbDatabase Events

In TTdbDatabase

OnPassword

Abgeleitet von TCustomConnection

(In der Embarcadero Dokumentation finden Sie weitere Informationen)

1.2.3.144 TTdbDatabase.BlobBlockSize

Legt die Blockgröße der Blob-Objekte für die nächste zu erstellende Tabelle fest

Delphi syntax:

```
property BlobBlockSize: Integer;
```

C++ syntax:

```
__property int BlobBlockSize = {read=FBlobBlockSize, write
=FBlobBlockSize, nodefault};
```

Beschreibung

Die Blockgröße ist die kleinste Speichereinheit für Blob-Objekte. TurboDB verwendet einen angemessenen Default-Wert für diesen Parameter, den Sie aber jederzeit nach den Bedürfnissen Ihrer Anwendung verändern können. Soll eine Tabelle beispielsweise vornehmlich sehr große Bilder speichern, kann es sinnvoll sein, die Blockgröße auf 16 oder 64 K oder noch mehr zu erhöhen. Die eingestellte Größe gilt für alle nachfolgenden Aufrufe von *TTdbTable.CreateTable* und *TTdbTable.AlterTable*.

Sie können auch mit verschiedenen Werte experimentieren um die beste Blockgröße für die Performanz Ihrer Anwendung herauszufinden.

1.2.3.145 TTdbDatabase.Backup

Erstellt ein Backup der Datenbank.

Delphi Syntax:

```
procedure Backup(TargetLocation: string; DatabaseType:
TTdbDatabaseType);
```

C++ Syntax:

```
void __fastcall Backup(const AnsiString TargetLocation, TTdbDatabaseType
DatabaseType);
```

Beschreibung

Der Typ der Backup-Datenbank ist unabhängig von der originalen Datenbank. Der Typ hängt ausschließlich vom Wert des Parameters *DatabaseType* ab. Das Ziel des Backups - Zielverzeichnis oder Zieldatei je nach Typ - muss nicht existieren. Backup kann bei laufenden Betrieb aufgerufen werden. Es versucht einen konsistenten Zustand der Datenbank abzuwarten und erstellt eine Kopie an der angegebenen Stelle. Falls das in der in *LockingTimeout* angebenen Zeitspanne nicht möglich ist, wird eine Exception geworfen.

1.2.3.146 TTdbDatabase.AutocreateIndexes

Gibt an, ob eine Query während ihrer Ausführung temporäre Indexe anlegen darf.

Delphi Syntax:

```
property AutoCreateIndexes: Boolean;
```

C++ Syntax:

```
__property bool AutoCreateIndexes = {read=FExclusive, write
=SetExclusive, nodefault};
```

Beschreibung

Wenn AutoCreateIndexes auf *True* steht (default), kann eine Query temporäre Indexes selbstständig anlegen um die Ausführungsgeschwindigkeit zu erhöhen. Temporäre Indexe werden automatisch gelöscht, wenn die letzte Session den Zugriff auf die Datenbank beendet.

1.2.3.147 TTdbDatabase.CacheSize

Spezifiziert die Größe des Speichers, den die Datenbank als Cache verwendet.

Delphi Syntax:

```
property CacheSize: Integer;
```

C++ Syntax:

```
__property int CacheSize;
```

Beschreibung

Erhöhen Sie den Wert um eine bessere Performanz für Datenbank-Operationen zu erhalten. Verringern Sie den Wert um anderen Anwendungen mehr Speicher zu belassen.

1.2.3.148 TTdbDatabase.CloseCachedTables

Schließt alle nicht benötigten Datendateien physikalisch.

Delphi Syntax:

```
procedure CloseCachedTables;
```

C++ Syntax:

```
void __fastcall CloseCachedTables(void);
```

Beschreibung

Das Schließen eines Datasets führt nicht automatisch dazu, dass auch die zugehörigen Datendateien geschlossen werden. Dies erhöht die Zugriffsgeschwindigkeit, falls die Daten erneut benötigt werden.

Die Ausführung von *CloseCachedDatasets* schließt physikalisch alle mit Datenbank Komponente verknüpften und nicht benutzten Datendateien.

Diese Methode kann notwendig sein, wenn Dateioperationen auf den Datendateien ausgeführt werden sollen, zB. bei Backup/Wiederherstellung

1.2.3.149 TTdbDatabase.CloseDataSets

Schließt alle Datasets.

Delphi Syntax:

```
procedure CloseDataSets;
```

C++ Syntax:

```
void __fastcall CloseDataSets(void);
```

Beschreibung

CloseDataSets schließt alle Datasets die über diese Datenbank Komponente zusammengeschlossen sind.

1.2.3.150 TTdbDatabase.Commit

Beendet eine Transaktion und führt ein Commit für alle Änderungen gegen die Datenbank aus.

Delphi Syntax:

```
procedure Commit;
```

C++ syntax:

```
void __fastcall Commit(void);
```

Beschreibung

Verwenden Sie *Commit* um die in der aktuellen Transaktion durchgeführten Änderungen in der Datenbank zu speichern. Die Prozedur löst eine Exception aus, falls keine Transaktion mit StartTransaction eingeleitet wurde.

1.2.3.151 TTdbDatabase.Compress

Reduziert die Dateigröße einer Single-File-Datenbank auf das Minimum.

Delphi Syntax:

```
procedure Compress;
```

C++ Syntax:

```
void __fastcall Compress(void);
```

Beschreibung

In einer Single-File-Datenbank wird der Platz, der durch das Löschen von Datensätzen entsteht zwar wieder für neue Datensätze verwendet, aber nicht freigegeben. Dadurch wird die Dateigröße einer Single-File-Datenbank auch dann nicht kleiner, wenn alle Datensätze in allen Tabellen gelöscht werden. Dieses Verhalten ist optimal für schnelle Datenbank-Operationen, kann aber manchmal nachteilig sein, wenn man eine kleine Datenbank-Datei benötigt z.B. zur Auslieferung. Ein Aufruf von *Compress* gibt alle unbenutzten Reserven in der Datenbank-Datei frei.

Es muss absolut sichergestellt sein, dass beim Aufruf von *Compress* kein anderer Zugriff auf die Datenbank besteht. Der sicherste Weg dies zu gewährleisten ist die Datenbank zu schließen und in exklusiven Modus zu öffnen.

Die Methode ist bei Verzeichnis-Datenbank wirkungslos.

1.2.3.152 TTdbDatabase.ConnectionId

Bezeichnet die Id zur Identifikation dieser Datenbankverbindung.

Delphi Syntax:

```
property ConnectionId: Integer;
```

C++ Syntax:

```
__property unsigned ConnectionId = {read=FConnectionId, nodefault};
```

Beschreibung

Lesen Sie diese Eigenschaft um die Id der Datenbankverbindung zu erhalten, die Sie benutzen. Wenn eine TTdbDatabase einer Verbindung zur TurboDB Engine öffnet, wird eine eindeutige Id zugewiesen. Diese Connection Id wird in Lock-Dateien verwendet um die sperrende Datenbankverbindung zu identifizieren.

1.2.3.153 TTdbDatabase.ConnectionName

Bezeichnet einen lesbaren Namen für die Datenbank Verbindung

Delphi Syntax:

```
property ConnectionName: String;
```

C++ Syntax:

```
__property AnsiString ConnectionName = {read=FConnectionName, write
=FConnectionName};
```

Beschreibung

Verwenden Sie diese Eigenschaft um Ihrer TurboDB Connection einen lesbaren Namen für dieses Datenbankobjekt zu geben. Der Name ist z.B. hilfreich, wenn Sie eine Liste der aktuellen Anwender anzeigen möchten.

1.2.3.154 TTdbDatabase.DatabaseName

Weist der Datenbankverbindung einen Namen zu.

Delphi Syntax:

```
property DatabaseName: String;
```

C++ Syntax:

```
__property AnsiString DatabaseName = {read=FDatabaseName, write
=SetDatabaseName};
```

Beschreibung

Verwenden Sie *DatabaseName* um der Datenbank einen Namen zu geben. Sie können einen beliebigen Namen vergeben, in der Anwendung darf es aber keine weitere Datenbankverbindung mit diesem Namen geben. *DatabaseName* muss mit der Eigenschaft *DatabaseName* der *TTdbDataSet* Komponenten identisch sein, die diese Datenbankverbindung verwenden.

Hinweis

Der Versuch die Eigenschaft zu setzen während *Connected* gleich true ist, löst eine Exception aus.

1.2.3.155 TTdbDataBase.Exclusive

Ermöglicht einer Anwendung alleinigen Zugriff auf eine Datenbank.

Delphi Syntax:

```
property Exclusive: Boolean;
```

C++ Syntax:

```
__property bool Exclusive = {read=FExclusive, write=SetExclusive,
nodefault};
```

Beschreibung

Öffnet die Datenbank exklusiv. Anderen Anwendungen ist der Zugriff auf die Tabellen nicht möglich.

1.2.3.156 TTdbDatabase.FlushMode

Bezeichnet den Vorgabewert für den Flush Mode der Tabellen dieser Datenbank

Delphi Syntax:

```
property FlushMode: TTdbFlushMode;
```

C++ Syntax:

```
__property TTdbFlushMode FlushMode = {read=FFlushMode, write=FFlushMode,
nodefault};
```

Beschreibung

Setzen Sie diese Eigenschaft um den Flush Modus für alle Tabellen dieser Datenbank zu setzen. Der Wert *tfmDefault* ist identisch mit *tfmFast*.

1.2.3.157 TTdbDatabase.IndexPageSize

Legt die Seitengröße für den Index fest, der als nächstes erzeugt wird.

Delphi Syntax:

```
property IndexPageSize: Integer;
```

C++ Syntax:

```
__property int IndexPageSize = {read=FIndexPageSize, write
=FIndexPageSize, nodefault};
```

Beschreibung

Normalerweise berechnet TurboDB die Seitengröße für einen Index intern. Durch Setzen dieser Eigenschaft können Sie diesen Wert für Spezialfälle anpassen. Die Eigenschaft legt die Größe einer Bayer-Baum-Seite in Bytes fest. Der eingestellte Wert wird für alle nachfolgenden Aufrufe von *TTdbTable.AddIndex* verwendet.

Je kleiner die Seiten sind, desto höher wird der Index-Baum werden. Normalerweise sollten Sie die Größe einer Indexseite so wählen, dass zwischen 10 und 300 Datensätze pro Seite indiziert werden können. Im Zweifelsfall können Sie mit verschiedenen Größen experimentieren um die schnellste Variante für Ihre Anwendung herauszufinden.

1.2.3.158 TTdbDatabase.Location

Bezeichnet den Speicherplatz der Datenbank

Delphi Syntax:

```
property Location: String;
```

C++ Syntax:

```
__property TTdbFlushMode FlushMode = {read=FFlushMode, write=FFlushMode,
nodefault};
```

Beschreibung

Verwenden Sie *Location* um die zu verwendende TurboDB Datenbank zu spezifizieren. *Location* ist entweder ein Datenbank-Verzeichnis in dem die Tabellen standardmäßig gesucht werden, (sogenannte Verzeichnis Datenbank) oder den Namen einer TurboDB Single-File Datenbank Datei (Single-File Datenbank). Single-File Datenbanken haben die Dateierweiterung *tdbd*.

Die Verwendung von relativen Dateinamen (führender Punkt) ist nicht zugelassen und führt zu einer Exception.

Anmerkung

Zur Entwurfszeit können Sie mit einem Doppelklick auf eine *TTdbDatabase* Komponente den Komponenten-Editor öffnen und nach einem Datenbank Speicherplatz suchen.

1.2.3.159 TTdbDatabase.LockingTimeout

Legt die Wartezeit bei gesperrten Datensätzen fest.

Delphi Syntax:

```
property LockingTimeout: Integer;
```

C++ Syntax:

```
__property int LockingTimeout = {read=FLockingTimeout, write
=FLockingTimeout, nodefault};
```

Beschreibung

Wenn eine Session einen Datensatz bearbeiten möchte, der schon von einer anderen Session bearbeitet wird, wartet TurboDB die über diese Eigenschaft festgelegte Zeit, bevor die zweite Session eine Ausnahme wirft. Die Angabe ist in Millisekunden.

1.2.3.160 TTdbDatabase.OnPassword

Tritt auf falls eine Anwendung versucht eine geschützte TurboDB Tabelle zum ersten Mal zu öffnen.

Delphi Syntax:

```
TTdbPasswordEvent = procedure(Sender: TObject; const TableName: string;
var Key: WideString; var Retry: Boolean) of object;
```

```
property OnPassword: TTdbPasswordEvent;
```

C++ Syntax:

```
typedef void __fastcall (__closure *TTdbPasswordEvent)(System::TObject*
Sender, const AnsiString TableName, WideString &Key, bool &Retry);
```

```
__property TTdbPasswordEvent OnPassword = {read=FOnPassword, write
=FOnPassword};
```

Beschreibung

Schreiben Sie eine Ereignisbehandlungsroutine um eine spezielle Aktion durchzuführen wenn die Anwendung versucht eine passwortgeschützte Tabelle zum ersten Mal zu öffnen. Um den Zugriff auf die Tabelle zu ermöglichen, muss die Prozedur den Parameter *Key* setzen. Verwenden Sie *Retry*, um das Öffnen der Tabelle bei Bedarf zu verhindern. Wird für *Retry* True zurückgegeben, wird versucht die Tabelle mit dem neuen Passwort zu öffnen. Wird für *Retry* False zurückgegeben, werde der Versuch die Tabelle zu öffnen beendet.

Hinweis

Falls keine Ereignisbehandlungsroutine für *OnPassword* existiert, TurboDB aber unzureichende Zugriffsrechte erkennt, wird eine Exception ausgelöst.

Beispiel

Das folgende Beispiel zeigt eine typische Ereignisbehandlungsroutine und den Quelltext zum Öffnen der Tabelle.

```
procedure TForm1.Password(Sender: TObject; const TableName: string; var
Key: WideString; var Retry: Boolean);
begin
     Key := InputBox('Enter password', 'Password:', '');
     Retry := (Key > '');
end;
```

```
procedure TForm1.OpenTableBtnClick(Sender: TObject);
begin
     Database.OnPassword := Password;
     try
          Table1.Open;
     except
          if not Table1.Active then begin
               ShowMessage('Could not open table');
               Application.Terminate;
    end;
  end;
end;
```

1.2.3.161 TTdbDatabase.PrivateDir

Spezifiziert das Verzeichnis, in dem TurboDB temporäre Dateien für Tabellen ablegt, die über diese Datenbank zusammengefaßt sind.

Delphi Syntax:

```
property PrivateDir: String;
```

C++ Syntax:

```
__property AnsiString PrivateDir = {read=FPrivateDir, write
=FPrivateDir};
```

Beschreibung

Verwenden Sie *PrivateDir* um das Verzeichnis festzulegen, das TurboDB für temporäre Dateien verwendet, die bei der Bearbeitung von Datenbanktabellen erzeugt werden, wie zum Beispiel bei der Verarbeitung von SQL Abfragen.

Für gewöhnlich wird *PrivateDir* zur Laufzeit gesetzt, so dass die Festplatte des Anwenders zur temporären Ablage benutzt wird. Die lokale Speicherung dieser Dateien erhöht die Performanz.

Falls *PrivateDir* nicht explizit gesetzt wird, speichert TurboDB temporäre Dateien im Verzeichnis Temp unter Windows. Falls Sie die Eigenschaft setzen, stellen Sie sicher, dass das Verzeichnis nicht von mehreren Benutzern oder Anwendungen gleichzeitig als temporäre Ablage verwendet wird.

Hinweis

Bei Anwendungen, die direkt von einem Netzwerkserver gestartet werden, sollte die Anwendung *PrivateDir* auf ein Verzeichnis der lokalen Festplatte des Benutzers setzen. Zum einen, um die Geschwindigkeit zu erhöhen, zum anderen, um die temporären Dateien nicht auf dem Serverlaufwerk zu halten, wo Konflikte mit den temporären Dateien anderer Instanzen der Anwendung entstehen könnten.

1.2.3.162 TTdbDatabase.RefreshDataSets

Aktualisiert die Komponenten für den Datenzugriff deren Datenquelle von anderen Prozessen geändert wurde.

Delphi Syntax:

```
procedure RefreshDataSets;
```

C++ Syntax:

```
void __fastcall RefreshDataSets(void);
```

Beschreibung

Verwenden Sie *RefreshDataSets* um alle mit dieser Datenbank-Komponente verknüpften Datasets zu aktualisieren, die von anderen Anwendungen in einer Multi-User Umgebung geändert wurden. *RefreshDataSets* prüft an welchen Datenmengen seit dem letzten Update Änderungen vorgenommen wurden und aktualisiert diese.

Der typische Einsatz von *RefreshDataSets* ist in einem *OnTimer* Handler. Auf diese Weise kann in regelmäßigen Abständen eine Aktualisierung der Datasets vorgenommen werden um die durchgeführten Änderungen der Arbeitsgruppe sichtbar zu machen.

1.2.3.163 TTdbDatabase.Rollback

Bricht eine Transaktion ab und verwirft die gemachten Änderungen.

Delphi Syntax:

```
procedure Rollback;
```

C++ Syntax:

```
void __fastcall Rollback(void);
```

Beschreibung

Verwenden Sie *Rollback* um alle Modifikationen der aktuellen Transaktion zurückzunehmen. Die Prozedur löst eine Exception aus, falls vorher keine Transaktion mit dem Aufruf von *StartTransaction* eingeleitet wurde.

1.2.3.164 TTdbDatabase.StartTransaction

Startet eine Transaktion.

Delphi Syntax:

```
procedure StartTransaction;
```

C++ Syntax:

```
void __fastcall StartTransaction(void);
```

Beschreibung

Verwenden Sie *StartTransaction* um eine Transaktion mit dem Isolationslevel Read Committed zu starten. Da TurboDB nur eine aktive Transaktion pro Datenbanksession haben kann, löst der Aufruf der Methode eine Exception aus, falls bereits eine Transaktion läuft. Durch den Aufruf von *Commit* oder *Rollback* wird die Transaktion beendet.

1.2.3.165 TTdbEnumValueSet

TTdbEnumValueSet ermöglicht den komfortablen Zugriff auf die Fähigkeiten eines Aufzählungstyps (*dtEnum*).

Unit

TdbEnumValueSet

Beschreibung

Verwenden Sie *TTdbEnumValueSet* um

- eine Kombobox zur Verfügung zu stellen, die die Werte des Auswahlfeldes zur Eingabe in Formularen und Grids bereitstellt
- Aliase für die Werte des Auswahlfeldes zu definieren (z.B. um Ihre Applikation in eine andere Sprache zu übersetzen)

TTdbEnumValueSet ist von *TDataSet* abgeleitet um mit Lookupfelder zusammenarbeiten zu können. Daher können Sie Auswahlfelder mit jeder Datenzugriff-Komponente verwenden, das die Lookup Technik unterstützt.

1.2.3.166 TTdbEnumValueSet Hierarchy

Hierarchie

TObject
|
TPersistent
|
TComponent
|
TDataSet
|
TTdbEnumValueSet

1.2.3.167 TTdbEnumValueSet Properties

In TTdbEnumValueSet

DataSource

EnumField

Values

Abgeleitet von TDataSet

(In der Embarcadero Dokumentation finden Sie weitere Informationen)

1.2.3.168 TTdbEnumValueSet.DataSource

Identifiziert die Datenquelle des Auswahlfeldes.

Delphi Syntax:

```
property DataSource: TDataSource;
```

C++ Syntax:

```
__property Db::TDataSource* MasterSource = {read=FMasterSource, write
=SetDataSource};
```

Beschreibung

Setzen Sie *DataSource* um das EnumValueSet mit der TdbDataSet Datenquelle zu verbinden, die das Auswahlfeld beinhaltet.

1.2.3.169 TTdbEnumValueSet.EnumField

Die Eigenschaft legt das Auswahlfeld der Datenquelle fest.

Delphi Syntax:

```
property EnumField: String;
```

C++ Syntax:

```
__property AnsiString EnumField = {read=GetDataField, write
=SetDataField};
```

Beschreibung

Setzen Sie *EnumField* um das Auswahlfeld in der Datenquelle zu identifizieren.

1.2.3.170 TTdbEnumValueSet.Values

Liste der Werte und Aliase des Auswahlfeldes.

Delphi Syntax:

```
property Values: TStrings;
```

C++ Syntax:

```
__property Classes::TStrings* Values = {read=FItems, write=SetValues};
```

Beschreibung

Sie können *Values* benutzen, um Aliase für die Werte des Auswahlfeldes zu definieren. Die Aliasnamen werden anstelle der richtigen Werte angezeigt und können über die Eigenschaft *Values* übersetzt oder verändert werden.

1.2.3.171 TTdbQuery

TTdbQuery führt SQL Statements auf der TurboDB Engine aus und ermöglicht den Zugriff auf die Ergebnismengen.

Unit

TdbQuery

Beschreibung

Verwenden Sie *TTdbQuery* um eine Ergebnismenge von einer oder mehreren Datenbanktabellen zu erzeugen, indem Sie ein SQL Select Statement verwenden oder um eine Datenbanktabelle via INSERT, DELETE oder UPDATE Statements zu editieren. Die verwendete SQL Syntax finden Sie im Turbo SQL Guide.

Anmerkung

TTdbQuery ist nur in der TurboDB Professional Edition enthalten.

1.2.3.172 TTdbQuery Hierarchy

Hierarchie

TObject

|

TPersistent

|

TComponent

|

TDataSet

|

TTdbDataSet

|

TTdbQuery

1.2.3.173 TTdbQuery Events

Abgeleitet von TTdbDataSet

OnProgress

OnResolveLink

Abgeleitet von TDataSet

(In der Embarcadero Dokumentation finden Sie weitere Informationen)

1.2.3.174 TTdbQuery Methods

In TTdbQuery

ExecSQL

Prepare

UnPrepare

Abgeleitet von TTdbDataSet

GetEnumValue

Locate

Lookup

Replace

Abgeleitet von TDataSet

(In der Embarcadero Dokumentation finden Sie weitere Informationen)

1.2.3.175 TTdbQuery Properties

In TTdbQuery

Params

RequestStable

SQL

SQLW

UniDirectional

UnPrepare

Abgeleitet von TTdbDataSet

FieldDefsTdb

Filter

Filtered

Version

Abgeleitet von TDataSet

(In der Embarcadero Dokumentation finden Sie weitere Informationen)

1.2.3.176 TTdbQuery.ExecSQL

Mit der Methode *ExecSQL* können Sie die SQL-Anweisung der Abfrage ausführen.

Delphi Syntax:

```
procedure ExecSQL;
```

C++ Syntax:

```
void __fastcall ExecSQL(void);
```

Beschreibung

Mit *ExecSQL* können Sie die Anweisung ausführen, die aktuell in der Eigenschaft SQL enthalten ist. Mit *ExecSQL* können Sie Anweisungen ausführen, die keinen Cursor auf Daten (wie z.B. INSERT, UPDATE, DELETE oder CREATE TABLE) zurückgeben.

Hinweis

Für die SELECT-Anweisung verwenden Sie statt *ExecSQL* die Methode *Open*.

Mit *ExecSQL* können Sie die Anweisung in der Eigenschaft SQL auf deren Ausführung vorbereiten, falls dies noch nicht geschehen ist. Um die Leistung zu erhöhen, sollte vor dem ersten Aufruf von *ExecSQL* die Methode *Prepare* aufgerufen werden.

1.2.3.177 TTdbQuery.Params

Die Eigenschaft *Params* enthält die Parameter für die SQL-Anweisung einer Abfrage.

Delphi Syntax:

```
property Params[Index: Word]: TParams;
```

C++ Syntax:

```
__property Db::TParams* Params = {read=FParams, write=SetParamsList,
stored=false};
```

Beschreibung

Mit *Params* können Sie die Namen, Werte und Datentypen der Parameter zur Laufzeit dynamisch abrufen und einstellen. Um die Parameter während des Entwurfs einzustellen, verwenden Sie den Kollektionseditor der Eigenschaft *Params*. *Params* ist ein null-basiertes Array mit *TParams* -Parameterdatensätzen. Index gibt das Element des Arrays an, auf das zugegriffen wird.

Hinweis

Sind die Namen aller Parameter bekannt, können ihre Werte mit *ParamByName* einfacher eingestellt und abgerufen werden. Mit *ParamByName* können Sie jedoch nicht den Datentyp oder den Namen des Parameters ändern.

Parameter, die in SELECT-Anweisungen verwendet werden, können nicht NULL sein. Parameter bei UPDATE- und INSERT-Anweisungen können dagegen NULL sein.

1.2.3.178 TTdbQuery.Prepare

Mit der Methode *Prepare* können Sie eine Abfrage an TurboDB übermitteln, damit diese vor der Ausführung optimiert wird.

Delphi Syntax:

```
procedure Prepare;
```

C++ Syntax:

```
void __fastcall Prepare(void);
```

Beschreibung

Mit *Prepare* können Sie Ressourcen für die Abfrage reservieren und zusätzliche Optimierungen durchführen. Wenn Sie vor der Ausführung einer Abfrage *Prepare* aufrufen, können Sie die Leistung einer Anwendung erhöhen.

In Delphi wird eine Abfrage automatisch für die Ausführung vorbereitet. Nach der Ausführung gibt Delphi die Ressourcen frei, die der Abfrage zugewiesen waren. Wenn eine Abfrage wiederholt ausgeführt werden soll, sollte sie immer explizit vorbereitet werden.

Für die Vorbereitung einer Abfrage werden Datenbankressourcen benötigt. Aus diesem Grund sollte die Vorbereitung einer Abfrage aufgehoben werden, nachdem die Abfrage ausgeführt wurde. Um die Vorbereitung einer Abfrage aufzuheben, verwenden Sie die Methode *UnPrepare*.

Hinweis

Wenn Sie den Text einer Abfrage zur Laufzeit ändern, wird die Abfrage automatisch geschlossen, und die Vorbereitung wird automatisch aufgehoben.

1.2.3.179 TTdbQuery.RequestStable

Fordert eine Ergebnismenge an, die sich nicht ändert wenn Zeilen editiert werden.

Delphi Syntax:

```
property RequestStable: Boolean;
```

C++ Syntax:

```
__property bool RequestStable = {read=FRequestStable, write
=FRequestStable, nodefault};
```

Beschreibung

Falls möglich erzeugt die Ausführung der Abfrage eine stabile Ergebnismenge, die nicht umsortiert oder eingeschränkt wird, wenn Zeilen bearbeitet werden. Wenn zum Beispiel ein Wert geändert wird, der zur Sortierordnung beiträgt, wird die geänderte Zeile nicht an eine andere Position verschoben selbst wenn die Sortierordnung jetzt verletzt ist.

1.2.3.180 TTdbQuery.SQL

Die Eigenschaft *SQL* enthält den Text der SQL-Anweisung für die Abfrage.

Delphi Syntax:

```
property SQL: TStrings;
```

C++ Syntax:

```
__property Classes::TStrings* SQL = {read=FSQL, write=SetQuery};
```

Beschreibung

SQL ermöglicht den Zugriff auf das SQL Statement kompatibel mit dem Objekt Inspektor, mit Embarcadero TQuery und mit früheren Versionen von TurboDB. Verwenden Sie *SQLW* falls Ihr SQL Statement Unicode Zeichen enthält oder falls Sie nicht mit *TStrings* arbeiten wollen.

1.2.3.181 TTdbQuery.SQLW

Enthält den Unicode Text eines SQL Statements zur Ausführung einer Query.

Delphi Syntax:

```
property SQLW: WideString;
```

C++ Syntax:

```
__property WideString SQLW = {read=FSQLW, write=SetQueryW};
```

Beschreibung

Mit *SQLW* können Sie die SQL-Anweisung angeben, die beim Aufruf der Methoden *ExecSQL* oder *Open* einer Abfragekomponente ausgeführt wird. Während des Entwurfs kann die Eigenschaft *SQLW* bearbeitet werden, indem im Objekt Inspektor der SQLBuilder aufgerufen wird.

Die Eigenschaft *SQLW* kann eine oder mehrere durch Semikolon getrennte SQL-Anweisungen enthalten. Liefert eine Stapelanweisung mehrere Ergebnismengen, wird nur die zurückgegeben.

Die *SQLW* Eigenschaft kann benutzt werden um mit Turbo SQL auf TurboDB Tabellen zuzugreifen. Die erlaubte Syntax ist eine SQL-92 Untermenge.

Die in die Eigenschaft *SQLW* eingefügte SQL-Anweisung kann entsprechend den Standardkonventionen von SQL-92 Parameter enthalten. Die Parameter werden in der Eigenschaft Params erstellt und gespeichert.

1.2.3.182 TTdbQuery.UniDirectional

Mit der Eigenschaft *UniDirectional* können Sie bestimmen, ob der bidirektionale Cursor der TurboDB Engine für die Ergebnismenge einer Abfrage aktiviert ist.

Delphi Syntax:

```
property UniDirectional: Boolean;
```

C++ Syntax:

```
__property bool UniDirectional = {read=FUniDirectional, write
=FUniDirectional, nodefault};
```

Beschreibung

Da TurboDB Cursor immer bidirektional sind, wird der Wert dieser Eigenschaft nicht berücksichtigt. Die Eigenschaft dient ausschließlich der BDE-Kompatibilität.

1.2.3.183 TTdbQuery.UnPrepare

Mit der Methode *UnPrepare* können Sie die Ressourcen freigeben, die für die Vorbereitung einer Abfrage zugewiesen wurden.

Delphi Syntax:

```
procedure UnPrepare;
```

C++ Syntax:

```
void __fastcall UnPrepare(void);
```

Beschreibung

Mit *UnPrepare* können Sie die Ressourcen freigeben, die für die Vorbereitung einer Abfrage zugewiesen wurden.

Für die Vorbereitung einer Abfrage werden Datenbankressourcen benötigt. Aus diesem Grund sollte die Vorbereitung einer Abfrage aufgehoben werden, nachdem die Abfrage ausgeführt wurde. Um die Vorbereitung einer Abfrage aufzuheben, verwenden Sie die Methode *UnPrepare*.

Hinweis

Wenn Sie den Text einer Abfrage zur Laufzeit ändern, wird die Abfrage automatisch geschlossen, und die Vorbereitung wird automatisch aufgehoben.

1.2.3.184 TTdbFieldDef

Das *TTdbFieldDef*-Objekt ist eine Felddefinition, die einem physischen Feld eines Datensatzes in einer der Datenmenge zugrundeliegenden TurboDB-Tabelle entspricht.

Unit

TdbDataSet

Beschreibung

Ein *TTdbFieldDef*-Objekt enthält die Definition eines Feldes in einer Tabelle. Die Felddefinition umfaßt folgende Attribute: den Namen des Feldes, den Datentyp und die Feldgröße. *TTdbFieldDef* -Objekte werden in der Regel in Kollektionen von Objekten, wie z.B. der Eigenschaft *FieldDefsTdb* der Komponente *TTdbDataSet* eingesetzt.

Wenn Sie mit einer vorhandenen Tabelle arbeiten, wird für jedes Feld einer Datenmenge automatisch eine Felddefinition erstellt. Mit Hilfe der Eigenschaften von *TTdbFieldDef* können Sie Informationen über bestimmte Felder in der Datenmenge ermitteln.

Beim Erzeugen einer neuer Tabelle z.B. mit der Methode *TTdbTable.CreateTable*, liefern *TTdbFieldDef*-Objekte die Definitionen für die neuen Felder, aus denen die neue Tabelle zusammengestellt wird.

Eine Felddefinition hat ein entsprechendes *TField*-Objekt, aber nicht alle *TField*-Objekte besitzen ein zugehöriges Felddefinitionsobjekt. Zum Beispiel haben berechnete Felder keine Felddefinitionsobjekte.

Es gibt zwei wesentliche Einsatzbereiche von *TTdbFieldDef*-Objekten:

- Sammeln von Information über die Feldtypen einer nicht geöffneten Tabelle.
- Festlegen von Felddefinitionen für eine neue Tabelle.

1.2.3.185 TTdbFieldDef Hierarchy

Hierarchie

TObject

|

TPersistent

|

TCollectionItem

|

TTdbFieldDef

1.2.3.186 TTdbFieldDef Properties

In TTdbFieldDefs

Expression

FieldNo

InitialFieldNo

InternalCalcField

DataTypeTdb

Specification

Abgeleitet von TDefCollection

(In der Embarcadero Dokumentation finden Sie weitere Informationen)

1.2.3.187 TTdbFieldDef Methods

In TTdbFieldDef

Assign

Abgeleitet von TCollectionItem

(In der Embarcadero Dokumentation finden Sie weitere Informationen)

1.2.3.188 TTdbFieldDef.Assign

Die Methode *Assign* kopiert den Inhalt einer anderen Felddefinition.

Delphi Syntax:

```
procedure Assign(Source: TPersistent); override;
```

C++ Syntax:

```
virtual void __fastcall Assign(Classes::TPersistent* Source);
```

Beschreibung

Mit *Assign* können Sie den Inhalt einer anderen *TTdbFieldDef* Instanz oder einer *TFieldDef* Instanz in das *TTdbFieldDef* Objekt kopieren, das die Methode ausführt.

1.2.3.189 TTdbFieldDef.DataTypeTdb

Der native Datentyp des physischen Feldes.

Delphi Syntax:

```
type TTdbDataType = (dtUnknown, dtString, dtInteger, dtSmallInt,
dtFloat, dtByte,   dtBoolean, dtDate, dtMemo, dtEnum, dtAutoInc, dtLink,
dtBlob, dtTime,   dtRelation, dtDateTime, dtWideString, dtWideMemo,
dtLargeInt, dtGuid); // dtGUID erst ab Delphi 5 oder höher
```

```
property DataTypeTdb: TTdbDataType;
```

C++ Syntax:

```
enum TTdbDataType {dtUnknown, dtString, dtInteger, dtSmallInt, dtFloat,
dtByte, dtBoolean, dtDate, dtMemo, dtEnum, dtAutoInc, dtLink, dtBlob,
dtTime, dtRelation, dtDateTime, dtWideString, dtWideMemo, dtLargeInt,
dtGuid};  // dtGUID erst ab C++ Builder 5 oder höher
```

```
__property TTdbDataType DataTypeTdb = {read=GetDataTypeTdb, write
=SetDataTypeTdb, nodefault};
```

Mit der Eigenschaft *DataTypeTdb* kann ermittelt werden, welchen nativen Datentyp ein Feld enthält.

Setzen Sie beim Hinzufügen von Felddefinitionen zu einer Datenmenge die Eigenschaft *DataTypeTdb*, um festzulegen, was für ein Feldtyp definiert wird.

1.2.3.190 TTdbFieldDef.CalcExpression

Beinhaltet einen Ausdruck zur Berechnung eines Feldes.

Delphi Syntax:

```
property CalcExpression: String;
```

C++ Syntax:

```
__property AnsiString CalcExpression = {read=FExpression, write
=FExpression};
```

Beschreibung

Verwenden Sie *CalcExpression*, um ein berechnetes Feld in einer TurboDB Tabelle zu definieren. Die Eigenschaft *InternalCalcField* entscheidet, ob die Berechnung nur bei neuen Datensätzen oder permanent durchgeführt wird.

Falls *InternalCalcField* True ist, wird er Feld-Wert immer neu berechnet und in der Tabelle gespeichert, wenn sich der Inhalt des Datensatzes ändert. Für *InternalCalcField* = False wird der Ausdruck zur Initialisierung des Feld-Wertes verwendet, der anschließend editiert werden kann.

Beispiel

Im folgenden wird eine Felddefinition zur Berechnung des Produktpreises inklusive 16% Mehrwertsteuer hinzugefügt:

```
with TdbTable1.FieldDefsTdb.Add('Price', dtFloat) do begin
     InternalCalcField := True;
     Expression := 'NetPrice * 1.16';
end;
```

Siehe auch

TTdbFieldDef.InternalCalcField

1.2.3.191 TTdbFieldDef.FieldNo

Mit der Eigenschaft FieldNo wird die physische Feldnummer zur Identifizierung des Feldes festgelegt.

Delphi Syntax:

```
property FieldNo: Integer;
```

C++ Syntax:

```
__property int FieldNo = {read=GetFieldNo, write=SetFieldNo, nodefault};
```

Beschreibung

FieldNo gibt an, an welcher physikalischen Position sich das Feld befindet, das die Felddefinition referenziert. Wenn der Wert von FieldNo beispielsweise 2 beträgt, dann handelt es sich hierbei um das zweite Feld der Tabellenstruktur.

Setzen Sie FieldNo um zu bestimmen an welcher Stelle ein Feld erzeugt wird wenn Sie eine Felddefinitionen zu einem DataSet hinzufügen.

Der Unterschied zwischen FieldNo und InitialFieldNo ist, daß InitialFieldNo den aktuellen Status der zugrundeliegenden Tabelle widerspiegelt während FieldNo die Position angibt, an der sich das Feld nach dem nächsten Aufruf von CreateTable oder AlterTable befinden wird.

1.2.3.192 TTdbFieldDef.InitialFieldNo

Beinhaltet die Position des physikalischen Feldes in der Tabelle.

Delphi Syntax:

```
property InitialFieldNo: Integer;
```

C++ Syntax:

```
__property short InitialFieldNo = {read=FInitialFieldNo, write
=FInitialFieldNo, nodefault};
```

Beschreibung

InitialFieldNo kann nur gelesen werden und wird intern verwendet, um Spaltenverschiebungen innerhalb der Felddefinitionen nachvollziehen zu können.

1.2.3.193 TTdbDatabase.InternalCalcField

Legt fest ob der Ausdruck in *CalcExpression* zur Berechnung eines Vorgabewerts oder für eine berechnete Spalte verwendet werden soll.

Delphi Syntax:

```
property InternalCalcField: Boolean;
```

C++ Syntax:

```
__property bool InternalCalcField = {read=FInternalCalcField, write
=FInternalCalcField, nodefault};
```

Beschreibung

Setzen Sie *InternalCalcField* auf True falls der Ausdruck in *CalcExpression* jedesmal berechnet werden soll, wenn sich der Datensatz ändert. Als Ergebnis wird die Spalte immer den aus *CalcExpression* resultierenden Wert haben. *InternalCalcField* ist auf *False zu setzen,* falls das Ergebnis aus CalcExpression als Vorgabewert für neue Datensätze dienen soll, der später nicht mehr neu berechnet wird.

1.2.3.194 TTdbFieldDef.Specification

Beinhaltet optionale Informationen zum Feldtyp.

Delphi Syntax:

```
property Specification: String;
```

C++ Syntax:

```
__property AnsiString Specification = {read=FSpec, write=FSpec};
```

Beschreibung

Je nach Feldtyp hat *Specification* eine unterschiedliche Bedeutung:

- In textuellen Feldern (String, WideString, Memo, WideMemo), steht der Name der Kollation in *Specification* Ein Leerstring bedeutet die Kollation *TurboDB*.
- In Auswahlfeldern beinhaltet *Specification* die Werte des Auswahlfeldes als komma-separierte Zeichenkette.
- Für Link und Relationsfelder definiert *Specification* den Name der verknüpften Tabelle.
- Felddefinitionen für automatische Zählerfelder (dtAutoInc) speichern die Anzeigeinformation in dieser Eigenschaft. Diese Information besteht aus einem einzelnen

Spaltennamen oder einer komma-separierten Liste von Spaltennamen der Tabelle und wird als lesbare Referenz auf den via Link- oder Relationsfeld verknüpften Datensatz angezeigt (Siehe Die Automatic Data Link Technologie)

1.2.3.195 TTdbFieldDefs

Ein *TTdbFieldDefs*-Objekt speichert die Felddefinitionsobjekte (*TTdbFieldDef*), die die physischen Felder der zugrundeliegenden Datenmenge repräsentieren.

Unit

TdbDataSet

Beschreibung

TTdbFieldDefs wird von einer Datenmenge für die Verwaltung der Felddefinitionen benutzt, die für die Erstellung von Feldobjekten verwendet werden, die Feldern in der Datenbanktabelle entsprechen. *TTdbTable* verwendet *TTdbFieldDefs*-Objekte bei der Erstellung einer neuen Datenbanktabelle.

Die Eigenschaften und Methoden von *TTdbFieldDefs* können für folgende Aufgaben eingesetzt werden:

- Zugreifen auf eine bestimmte Felddefinition
- Hinzufügen oder Löschen von Felddefinitionen in der Liste (neue Tabellen)
- Ermitteln der Anzahl definierter Felder
- Kopieren von Felddefinitionen in eine andere Tabelle

1.2.3.196 TTdbFieldDefs Hierarchy

Hierarchie

TObject

|

TPersistent

|

TCollection

|

TOwnedCollection

|

TDefCollection

|

TTdbFieldDefs

1.2.3.197 TTdbFieldDefs Methods

In TTdbFieldDefs

Add

Assign

Find

Abgeleitet von TDefCollection

(In der Embarcadero Dokumentation finden Sie weitere Informationen)

1.2.3.198 TTdbFieldDefs Properties

In TTdbFieldDefs

Items

Abgeleitet von TDefCollection

(In der Embarcadero Dokumentation finden Sie weitere Informationen)

1.2.3.199 TTdbFieldDefs.Add

Die Eigenschaft Add erstellt ein neues Felddefinitionsobjekt und fügt es der Eigenschaft Items dieses *TTdbFieldDefs*-Objekts hinzu.

Delphi Syntax:

```
function Add(const Name: String; DataTypeTdb: TTdbDataType; Size:
Integer = 0; Required: Boolean = False): TTdbFieldDef;
```

C++ Syntax:

```
HIDESBASE TTdbFieldDef* __fastcall Add(const AnsiString Name,
TTdbDataType DataTypeTdb, int Size = 0, bool Required = false, const
AnsiString Specification);
```

Beschreibung

Verwenden Sie *Add* um eine neue TdbFieldDef zu der Liste der Felddefinitionen hinzuzufügen. Add gibt eine Referenz auf das neue TdbFielDef Objekt zurück.

Add verwendet die übergebenen Werte in *Name*, *DataType*, *Size* und *Required* und weist Sie den entsprechenden Eigenschaften der neuen Felddefinition zu.

Falls bereits eine Felddefinition mit diesem Namen existiert wird ein EDatabaseError ausgelöst.

Beispiel

Der folgende Code leert die FieldDefTdb Liste und fügt drei Felder ein:

```
// Leeren der Liste der Felddefinitionen
TdbTable1.FieldDefsTdb.Clear;
// Ein Stringfeld der Laenge 40 hinzufuegen
TdbTable1.FieldDefsTdb.Add('Name', dtString, 40);
{ Ein Auswahlfeld mit den Werten "Sales", "Marketing", "Development"
hinzufuegen }
with TdbTable1.FieldDefsTdb.Add('Department', dtEnum) do Specification
:= 'Sales,Marketing,Development';
{ Ein Zaehlerfeld hinzufuegen, das als RecordId fuer den Datensatz
verwendet wird. Referenzen auf Datesaetze dieser Tabelle zeigen den
Namen der Person an. }
with TdbTable1.FieldDefsTdb.Add('RecordId', dtAutoInc) do Specification
:= 'Name';
{ Die Tabelle mit den neuen Felddefinitionen restrukturieren. }
TdbTable1.AlterTable;
```

Siehe auch

TTdbFieldDef

1.2.3.200 TTdbFieldDefs.Assign

Kopiert den Inhalt einer anderen Felddefinitionsliste in das aufrufende Objekt.

Delphi Syntax:

```
procedure Assign(Source: TPersistent); override;
```

C++ Syntax:

```
virtual void __fastcall Assign(Classes::TPersistent* Source);
```

Beschreibung

Verwenden Sie *Assign,* um den Inhalt einer TTdbFieldDefs Instanz in eine andere zu kopieren. Die *Assign* Methode entfernt alle Einträge in der Zielliste (das Objekt das die Methode ausführt), dann wird eine Kopie eines jeden Eintrags der Quellliste eingefügt. Sie können einer TdbFieldDefs Instanz entweder ein TdbFieldDefs oder ein FieldDefs Objekt zuweisen. *Assign* wird in erster Linie zum Kopieren bestehender Tabellenstrukturen eingesetzt.

1.2.3.201 TTdbFieldDefs.Find

Die Methode *Find* sucht mit Hilfe des Feldnamens eine Felddefinition im Array der Eigenschaft Items.

Delphi Syntax:

```
function Find(const AName: string): TTdbFieldDef;
```

C++ Syntax:

```
TTdbFieldDef* __fastcall Find(const AnsiString Name);
```

Beschreibung

Mit *Find* können Sie Informationen über eine bestimmte Felddefinition abrufen. Geben Sie den Feldnamen als Wert für den Parameter Name an.

1.2.3.202 TTdbFieldDefs.Items

Die Eigenschaft *Items* listet die nativen Felddefinitionen auf, die jedes physische Feld einer Datenmenge beschreiben.

Delphi Syntax:

```
property Items[Index: Integer]: TTdbFieldDef; default;
```

C++ Syntax:

```
__property TTdbFieldDef* Items[int Index] = {read=GetFieldDef, write
=SetFieldDef};
```

Beschreibung

Mit *Items* kann auf eine bestimmte Felddefinition zugegriffen werden. Mit dem Parameter *Index* wird die Felddefinition angegeben, auf die zugegriffen werden soll. *Index* ist ein Integer, der die Position der Felddefinition in der Liste der Felddefinitionen kennzeichnet. Der Wert kann zwischen 0 und Count - 1 liegen.

Die Eigenschaft ist der VCL Eigenschaft *TFieldDefs.Items* sehr ähnlich, enthält allerdings TTdbFieldDef Objekte anstelle der *TFieldDef* Objekte.

1.2.3.203 TTdbFlushMode

Bezeichnet dir Art des Schreibpuffers.

Delphi Syntax:

```
TTdbFlushMode = (tfmDefault, tfmSecure, tfmFast);
```

C++ Syntax:

```
enum TTdbFlushMode {tfmDefault, tfmSecure, tfmFast};
```

Beschreibung

Diese Werte finden Anwendung in TTdbDatabase.FlushMode und TTdbTable.FlushMode.

1.2.3.204 TTdbLockType

Bezeichnet die Art der anzuwendenden Sperre.

Delphi Syntax:

```
TTdbLockType = (tltWriteLock, tltTotalLock);
```

C++ Syntax:

```
enum TTdbLockType {tltWriteLock, tltTotalLock};
```

Beschreibung

tltWriteLock Eine Schreibsperre hält andere Anwendungen vom Schreiben in die Tabelle ab.

Nutzen Sie diese Art der Sperre um sicherzustellen, dass eine andere Anwendung keine Tabelleänderung während Ihrer Leseoperation durchführt.

tltTotalLock Eine Totalsperre verhindert jeden Zugriff anderer Anwendungen auf die Tabelle.

Nutzen Sie diese Art der Sperre um sicherzustellen, dass eine andere Anwendung die Tabelle während Ihrer Schreiboperation nicht lesen kann.

1.2.3.205 TTdbBlobProvider Class

TTdbBlobProvider unterstützt die Darstellung von Bildern verschiedener Formate aus Blob-Feldern.

Unit

TdbExtComps

Beschreibung

Ist ein *TdbBlobProvider* mit der Blob-Spalte einer Datenquelle verknüpft, versucht er das Dateiformat des aktuellen Bildes zu erkennen und lädt es. *TdbBlobProvider* ist erweiterbar, so dass neue Bildformate hinzugefügt werden können, für die ein *TGraphic* Abkömmling existiert. *TdbBlobProvider* kann auch zum Schreiben der Bilder in die Datenbank und sogar zum Verknüpfen der Bilder mit Blob-Feldern verwendet werden.

Siehe auch

Images Demo Program

1.2.3.206 TTdbBlobProvider Hierarchy

Hierarchy

TObject

|

TPersistent

|

TComponent

|

TTdbBlobProvider

1.2.3.207 TTdbBlobProvider Events

Ereignisse in TTdbBlobProvider

OnReadGraphic

OnUnknownFormat

Abgeleitet von TComponent

(In der Embarcadero Dokumentation finden Sie weitere Informationen)

1.2.3.208 TTdbBlobProvider Methods

In *TTdbBlobProvider*

Create

Destroy

RegisterBlobFormat

CreateTextualBitmap

LoadBlob

SetBlobData

SetBlobLinkedFile

Abgeleitet von *TComponent*

(In der Embarcadero Dokumentation finden Sie weitere Informationen)

1.2.3.209 TTdbBlobProvider Properties

In *TTdbBlobProvider*

BlobIsEmbedded

BlobFormat

BlobSize

BlobFormatTag

BlobFormatName

Picture

BlobDataStream

LinkedBlobFileName

DataSource

FieldName

Abgeleitet von *TComponent*

(In der Embarcadero Dokumentation finden Sie weitere Informationen)

1.2.3.210 TTdbBlobProvider.BlobDataStream Property

Bezieht sich auf einen Stream,der Blobs lesen und schreiben kann.

Delphi Syntax:

```
property BlobDataStream: TStream read FBlobDataStream;
```

C++ Syntax:

```
__property TStream BlobDataStream = {read=FBlobDataStream};
```

Beschreibung

Verwenden Sie *BlobDataStream* um eingebettete Blobs aus oder in eine Datenbank zu lesen oder schreiben.

1.2.3.211 TTdbBlobProvider.BlobFormat Property

Gibt das Format des aktuellen Blobs an.

Delphi Syntax:

```
property BlobFormat: TTdbBlobFormat read FBlobFormat;
```

C++ Syntax:

```
__property TTdbBlobFormat BlobFormat = {read=FBlobFormat};
```

Beschreibung

Lesen Sie den Inhalt dieser Eigenschaft um den Typ des aktuellen Bildes zu erfahren.

Siehe auch

TTdbBlobProvider.BlobFormatName

1.2.3.212 TTdbBlobProvider.BlobFormatName Property

Liefert den Namen des Formats des aktuellen Bildes.

Delphi Syntax:

```
property BlobFormatName: string read FBlobFormatName;
```

C++ Syntax:

```
__property String BlobFormatName = {read=FBlobFormatName};
```

Beschreibung

Lesen Sie diese Eigenschaft um den Namen des aktuellen Bildformats zu erfahren.

Siehe auch

TTdbBlobProvider.BlobFormat

1.2.3.213 TTdbBlobProvider.BlobFormatTag Property

Liefert den Kurznamen des aktuellen Bildes.

Delphi Syntax:

```
property BlobFormatTag: string read FBlobFormatTag;
```

C++ Syntax:

```
__property String BlobFormatTag = {read=FBlobFormatTag};
```

Beschreibung

LEsen Sie diese Eigenschaftr um die Kurzbezeichnung des aktuellen Bildformats zu erfahren.

Siehe auch

TTdbBlobProvider.BlobFormatName

1.2.3.214 TTdbBlobProvider.BlobIsEmbedded Property

Zeigt an ob das aktuelle Blob eingebettet oder verknüpft ist.

Delphi Syntax:

```
property BlobIsEmbedded: Boolean read GetBlobIsEmbedded;
```

C++ Syntax:

```
__property bool BlobIsEmbedded = {read=GetBlobIsEmbedded};
```

1.2.3.215 TTdbBlobProvider.BlobSize Property

Liefert die Größe des aktuellen Blobs in Bytes.

Delphi Syntax:

```
property BlobSize: Int64 read GetBlobSize;
```

C++ Syntax:

```
__property int64 BlobSize = {read=GetBlobSize};
```

Beschreibung

Diese Eigenschaft enthält die Größe des aktuellen Blobs in Bytes.

1.2.3.216 TTdbBlobProvider.DeleteBlob

Löscht das aktuelle Blob.

Delphi Syntax:

```
procedure DeleteBlob;
```

C++ Syntax:

```
virtual void __fastcall DeleteBlob(void);
```

Beschreibung

Rufen Sie *DeleteBlob* auf, um das aktuelle Blob aus der Datenbank zu entfernen.

1.2.3.217 TTdbBlobProvider.Create Constructor

Erzeugt eine Instanz der *TTdbBlobProvider* Komponente.

Delphi Syntax:

```
constructor Create(AOwner: TComponent); override;
```

C++ Syntax:

```
virtual TTdbBlobProvider * __fastcall TTdbBlobProvider(Classes::TComponent * AOwner
```

Beschreibung

Rufen Sie *Create* auf um eine Instanz der Blob-Provider Komponente zur Laufzeit zu erstellen.

1.2.3.218 TTdbBlobProvider.CreateTextualBitmap Class Method

Erzeugt ein Bitmap das Text darstellt.

Delphi Syntax:

```
class function CreateTextualBitmap(const Text: string): TGraphic;
```

C++ Syntax:

```
virtual static TGraphic * __fastcall SetText(System::String Text);
```

Beschreibung

Verwenden Sie Use *CreateTextualBitmap* um ein Bitmap mit Text zu erzeugen, das anstelle eines Blobs angezeigt wird, das nicht dargestellt werden kann. Repräsentiert das Blob beispielsweise eine Musikdatei oder ein unbekanntes Bildformat, kann der Text den Anwender davon in Kenntnis setzen.

1.2.3.219 TTdbBlobProvider.DataSource Property

Definiert die Datenquelle für die Blobs

Delphi Syntax:

```
property DataSource: TDataSource read GetDataSource write SetDataSource;
```

C++ Syntax:

```
__property TDataSource * DataSource = {read=GetDataSource, write=SetDataSource};
```

Beschreibung

Der Blob-Provider ließt von und schreibt in die angegebene Datenquelle.

Siehe auch

FieldName property

1.2.3.220 TTdbBlobProvider.Destroy Destructor

Löscht die *TTdbBlobProvider* Instanz.

Delphi Syntax:

```
destructor Destroy; override;
```

C++ Syntax:

```
virtual void __fastcall ~TDataSet(void);
```

1.2.3.221 TTdbBlobProvider.FieldName Property

Definiert die Datenbank-Spalte des Blobs.

Delphi Syntax:

```
property FieldName: string read GetFieldName write SetFieldName;
```

C++ Syntax:

```
__property System::String FieldName = {read=GetFieldName, write=SetFieldName};
```

Beschreibung

Die Eigenschaft bestimmt, in welches Feld der Datenquelle das Blob geschrieben und gelesen wird.

Siehe auch

TTdbBlobProvider.DataSource

1.2.3.222 TTdbBlobProvider.LinkedBlobFileName Property

Liefert den Name der Datei im Falle eines verknüpften Blobs.

Delphi Syntax:

```
property LinkedBlobFileName: string read FLinkedBlobFileName;
```

C++ Syntax:

```
__property System::String LinkedBlobFileName = {read=FLinkedBlobFileName, write=FLi
```

Beschreibung

In dieser Eigenschaft steht der Dateiname des verknüpften Blobs.

Siehe auch

TTdbBlobProvider.DataSource

1.2.3.223 TTdbBlobProvider.LoadBlob Method

Lädt das aktuelle Blob.

Delphi Syntax:

```
procedure LoadBlob;
```

C++ Syntax:

```
virtual void __fastcall LoadBlob(void);
```

Beschreibung

Der Aufruf von *LoadBlob* lädt das aktuelle Blob und stellt es für den Zugriff über das *Picture* Property zur Verfügung. Normalerweise passiert das automatisch, wenn sich die Daten der Datenquelle ändern, mit der der Blob-Provider verknüpft ist.

Siehe auch

Picture Eigenschaft

1.2.3.224 TTdbBlobProvider.OnReadGraphic Event

Wird ausgelöst wenn ein Blob aus der Datenbank gelesen wird.

Delphi Syntax:

```
TTdbBlobProviderReadGraphicEvent = procedure(Marker: Integer; var Graphic: TGraphic
```

```
property OnReadGraphic: TTdbBlobProviderReadGraphicEvent read
FOnReadGraphic write FOnReadGraphic;
```

C++ Syntax:

```
typedef void __fastcall (__closure *TTdbBlobProviderReadGraphicEvent)(int Marker, T
```

```
__property TTdbBlobProviderReadGraphicEvent = {read=FOnReadGraphic,
write=FOnReadGraphic};
```

Beschreibung

Dieses Ereignis muss behandelt werden, wenn sich die Anwendung selbst um die Erzeugung der Grafik kümmern will.

1.2.3.225 TTdbBlobProvider.OnUnknownFormat Event

Tritt auf, wenn der Blob-Provider auf ein unregistriertes Format trifft.

Delphi Syntax:

```
TTdbBlobProviderUnknownFormatEvent = procedure(Marker: Integer) of object;

property OnUnknownFormat: TTdbBlobProviderUnknownFormatEvent read FOnUnknownFormat
```

C++ Syntax:

```
typedef void __fastcall (__closure *TTdbBlobProviderUnknownFormatEvent)(int Marker)

__property TTdbBlobProviderUnknownFormatEvent = {read=FOnUnknownFormat,
write=FOnUnknownFormat};
```

Beschreibung

Dieses Ereignis muss bearbeitet werden, wenn sich die Anwendung selbst um die Behandlung unbekannter Bildformate kümmern soll.

1.2.3.226 TTdbBlobProvider.Picture Property

Stellt das Bild des aktuellen Blobs bereit.

Delphi Syntax:

```
property Picture: TPicture read GetPicture write SetPicture;
```

C++ Syntax:

```
__property TPicture * Picture = {read=GetPicture, write=SetPicture};
```

Beschreibung

Die Methode *LoadBlob* lädt das Blob aus der Datenquelle und erzeugt ein Bild, das dann über die Eigenschaft *Picture* verfügbar ist.

Siehe auch

LoadBlob method

1.2.3.227 TTdbBlobProvider.RegisterBlobFormat Class Method

Registriert ein Bild Format.

Delphi Syntax:

```
class procedure RegisterBlobFormat(BitMask, Pattern: Int64; const Tag,
Name: string; Format: TTdbBlobFormat; GraphicClass: TGraphicClass);
```

C++ Syntax:

```
virtual static void __fastcall RegisterBlobFormat(int64 BitMask, int64
Pattern, System::String Tag, System::String Name, TGraphicClass *
GraphicClass);
```

Parameter

BitMask	64-Bit Maske um die Muster Bits aus den ersten acht Bytes der Blob-Daten zu extrahieren
Pattern	64-Bit Muster um die aus den Blob-Daten extrahierten Bit-Muster zu vergleichen
Tag	Kurzname für das Format, z.B. *bmp*, *wav*, *wmf*, *gif*, *png*, *jpg*
Name	Langer lesbarer Name für das Format, z.B. Windows Meta File
Format	Nummerischer Wert zur Identifikation des Blob-Formats, muss innerhalb der Tabelle eindeutig sein
GraphicClass	Klassen Objekt eines *TGraphic* Abkömmlings, der fähig ist das Format mit

	ReadData und *WriteData* zu verarbeiten

Beschreibung

Standardmäßig kann der Blob-Provider Bitmaps, Windows Meta Files und Wave Dateien extrahieren. Weitere Bildformate wie *gif*, *png*, *jpeg* etc. können durch Registrieren des Formats und einer geeigneten Grafik-Klasse hinzugefügt werden. Beim Laden eines Blobs führt der Blob-Provider eine bitweise And-Operation für die ersten vier Bytes des Blobs mit *BitMask* durch und Vergleicht das ERgebnis mit *Pattern*. Bei Übereinstimmung erzeugt er eine Instanz der Grafik-Klasse, verknüpft einen Stream zum Lesen des Blobs damit und ruft die *ReadData* Methode auf. Die resultierende Grafik wird dann an das Picture-Objekt der Eigenschaft *Picture* übergeben.

Der Blob-Provider liest die ersten acht Bytes der Blob-Daten als 64 Bit Integer Zahl, daher müssen die *Pattern* Bytes entgegengesetzt der physikalischen Reihenfolge angegeben werden, der Little-Endian-Konvention geschuldet.

Beispiel

Dieser Code registriert die GIF, JPEG und PNG Formate für den Blob-Provider

```
ImageBlobProvider.RegisterBlobFormat($ffff, $d8ff, 'JPG', 'JPEG Image', tbfJPG, TJp
ImageBlobProvider.RegisterBlobFormat($ffffffff, $38464947, 'GIF', 'GIF Image', tbfG
ImageBlobProvider.RegisterBlobFormat($ffffffffffffffff, $0a1a0a0d474e5089, 'PNG', '
```

Siehe auch

LoadBlob method
Picture property

1.2.3.228 TTdbBlobProvider.SetBlobData Method

Schreibt einen eingebetteten Blob

Delphi Syntax:

```
procedure SetBlobData(Stream: TStream; Format: TTdbBlobFormat);
```

C++ Syntax:

```
virtual static void __fastcall SetBlobData(TStream * Stream,
TTdbBlobFormat Format);
```

Beschreibung

Mit dieser Methode wird ein eingebetteter Blob in die Datenquelle geschrieben.

1.2.3.229 TTdbBlobProvider.SetBlobLinkedFile Method

Schreibt einen verknüpften Blob.

Delphi Syntax:

```
procedure SetBlobLinkedFile(const FileName: string);
```

C++ Syntax:

```
virtual static void __fastcall SetBlobLinkedFile(const System::String
FileName);
```

Beschreibung

Mit dieser Methode wird ein verknüpfter Blob in die Datenquelle geschrieben. Dabei wird lediglich der Dateipfad in die Datenbank geschrieben und der Blob-Provider liest die Blob-Daten aus der Datei.

1.3 Datenbank Engine

dataweb bietet derzeit zwei Datenbank Engines an. TurboDB Native läuft auf allen 32-Bit Windows Plattformen und TurboDB Managed, für das .NET Framework und .NET Compact Framework.

TurboDB verfügt über einen sehr schnellen und kompakten 32 Bit Datenbank Kernel, der im Laufe der letzten acht Jahre fortlaufend an die Bedürfnisse der Anwendungs-Entwicklern angepasst wurde. Es sind keine Konfigurationsarbeiten zu erledigen, so dass sich eine Installation auf einfaches Kopieren der Dateien beschränkt. Diese Eigenschaft macht TurboDB zur idealen Lösung für herunterladbare, CD-, DVD- oder Web-Anwendungen, die auf einem Webserver laufen sollen, der nur über FTP zu erreichen ist.

Dieses Buch beschreibt alle Features der TurboDB Datenbank Engine, die unabhängig von der verwendeten Entwicklungsumgebung und Komponenten-Bibliothek sind. Informationen zur VCL Komponenten-Bibliothek für Delphi und C++Builder finden Sie in "TurboDB 5 für VCL". Die Klassen des ADO.NET Providers für das .NET Framework sind in "TurboDB 5 für .NET" beschrieben.

Beide TurboDB Datenbank Engines zeichnen sich durch folgende Eigenschaften aus:

- Geringer Speicherbedarf
- Schnell
- Keine Installation
- Keine Konfiguration
- Mehrbenutzerfähigkeit
- Lizenz-freie Weitergabe erstellter Anwendungen
- Visueller Datenbank-Manager
- Viele weiter kostenfreie Werkzeuge

Die **Managed Engine** verfügt zusätzlich über folgende Vorteile:

- Läuft auf Windows für mobile Geräte: Pocket PC und Smartphone.
- Benötigt keine speziellen Rechte um in einer .NET Umgebung zu laufen (z.B. Unsafe Code)
- Unterstützt benutzerdefinierte Funktionen, Stored Procedures und benutzerdefinierte Aggregates.

Die **Native Engine** bietet diese Vorteile:

- Verschlüsselte Datenbank-Dateien
- Spezielle Spaltentypen für one-to-many und many-to-many Beziehungen zwischen Tabellen

1.3.1 Neuigkeiten und Upgrade

1.3.1.1 Neu in TurboDB Win32 v6

TurboDB 6 beinhaltet eine große Anzahl an neuen und erweiterten Features. Um einige dieser neuen Fähigkeiten nutzen zu können ist es nötig, bestehende Tabellen auf das neue Tabellen-Format 6 zu bringen. In den entsprechenden Kapiteln der Dokumentation sehen Sie, ob dies jeweils der Fall ist.

- Schnellere Verarbeitung von Unicode Zeichenketten
- Schnellere Berechnungen in allen Bereichen der Datenbank-Engine
- Optimiertes Speicher-Format für Index-Einträge mit variabler Länge. Insbesondere Indexe über lange String-Felder profitieren vom reduzierten Speicherbedarf und der schnelleren Verarbeitung.

- Präzise Ergebnis-Datentypen für berechnete Spalten in SQL
- Für String-Spalten stehen jetzt Collations zur Verfügung, die definieren wie die Zeichenketten zu sortieren und vergleichen sind.
- Die Dateierweiterungen für Datenbank-Objekte, wie Tabellen, Indexe usw. wurden für den neuen Tabellen-Level 6 geändert. Zur Vermeidung von Namenskonflikten mit anderen Anwendungen beginnen sie nun einheitlich mit *tdb*.
- Benutzerdefinierte Separatoren für die Volltextindizierung. Siehe CREATE FULLTEXTINDEX SQL Kommando.
- Selektive Volltext-Suche in SQL: Das Prädikat CONTAINS unterstützt nun auch die Definition einzelner Spalten, in denen das gesuchte Wort enthalten sein muss: *WHERE CONTAINS('aWord' in Column1, Column2, Column8)*
- Definition berechneter Spalten mit den Kommandos *ALTER TABLE* und *CREATE TABLE*

Siehe auch

Upgrade auf TurboDB Win32 v6

1.3.1.2 Upgrade auf TurboDB Win32 v6

Bei der Umstellung von TurboDB 5 auf TurboDB 6 sind folgende Dinge zu beachten:

Sprachtreiber

Sprachtreiber werden von TurboDB 6 nicht mehr unterstützt. Stattdessen können sowohl für Tabellen als auch für einzelne Spalten Kollationen definierte werden, um die alphabetische Sortierung festzulegen. Wenn Sie eine Tabelle, die einen Sprachtreiber benutzt, nach TurboDB 6 verbessern möchten, müssen Sie erst den Sprachtreiber mit TurboDB 5 entfernen, anschließend den Tabellenlevel mit TurboDB 6 auf Level 6 erhöhen und dabei die gewünschte Kollation zuweisen.

Sortierung und Vergleich von Zeichenketten

Zeichenketten werden jetzt entsprechend Ihrer Kollationen sortiert und verglichen. Für ältere Tabellen wird die Groß-Kleinschreibung bei String-Vergleichen nicht beachtet, da sich Sortier- und Vergleich-Operationen von TurboDB 5 unterscheiden. Falls dies nicht den Anforderungen entspricht, muss eine andere Kollation zugewiesen werden.

Wichtiger Hinweis

Beachten Sie bitte, dass TurboDB 6 und TurboDB 5 nicht gleichzeitig auf Datenbank Tabellen zugreifen können. Die Meldung, dass die Sperrdateien inkompatibel sind, wäre die Folge. Falls diese Meldung angezeigt wird obwohl keine TurboDB 5 Anwendung läuft, sind noch net/rnt/mov/rmv Dateien vorhanden. Löschen Sie diese Dateien und alles wird wie erwartet funktionieren.

Bei der Umstellung von TurboDB 4 auf TurboDB 5 sind einige Punkte zu beachten:

Wichtiger Hinweis

Beachten Sie bitte, dass TurboDB 5 und TurboDB 4 nicht gleichzeitig auf Datenbank Tabellen zugreifen können. Die Meldung "Error in log-in" wäre die Folge. Falls diese Meldung angezeigt wird obwohl keine TurboDB 4 Anwendung läuft, sind noch net/rnt/mov/rmv Dateien vorhanden. Löschen Sie diese Dateien und alles wird wie erwartet funktionieren.

Neu reservierte Schlüsselwörter

Die folgenden Bezeichner sind neue reservierte Schlüsselwörter in TurboDB 5. Falls Ihr Datenbank-Schema eines dieser Wörter als Tabellen- oder Spalten-Name verwendet, müssen sie diese in SQL-Statements entweder mit doppelten Anführungszeichen umschließen oder

umbenennen:

ACTION, ALL, ANY, CASCADE, CASE, CAST, DICTIONARY, ENCRYPTION, EXCEPT, EXISTS, FULLTEXTINDEX, INTERSECT, NO, SOME, TOP, UNION, WHEN

Verwendung von Anführungszeichen

TurboDB 5 verwendet einfache Anführungszeichen ausschließlich für Zeichenketten Literale. In TurboDB 4 waren doppelte Anführungszeichen hierfür ebenso erlaubt und müssen nun ausgetauscht werden. Doppelte Anführungszeichen sind nun ausschließlich für Bezeichner zu verwenden und ermöglichen es mit Namen zu arbeiten, die Leerzeichen oder Sonderzeichen beinhalten oder mit reservierten Schlüssel-Wörtern übereinstimmen.

Verschlüsselung

TurboDB bietet nun zusätzliche Methoden für eine sicher Verschlüsselung. Aus diesem Grund hat sich die Syntax für die Statements create table und alter table geändert.

Volltext Indexe

Die Volltext Indizierung wurde grundlegenden neu gestaltet und ist nun wesentlich schneller und dazu gewartet. Die alten Volltext Indexe werden für die alten Tabellen-Formate natürlich weiter unterstützt. Wenn Sie aber Ihre Tabellen upgraden, müssen Sie den Volltext-Index-Teil Ihrer SQL-Statements anpassen.

1.3.1.3 Neu in TurboDB Managed v2

Version 2.0

- Das neue Prädikat CONTAINS in Verbindung mit der neuen Volltext-Indizierung ermöglicht eine mächtige und sehr schnelle Suche nach beliebigen Wörtern.
- Tabellen-Namen können bis zu 79 Zeichen lang, Spaltennamen bis zu 128 Zeichen lang sein.
- Einführung eines Connection-Pools zur Verbesserung der Performanz.
- Unterstützung von Fast Encryption und Rijndael (AES).
- Mit der Klasse *TurboDBConnection* wird die Komprimierung der Datenbank möglich
- UNION, INTERSECT und EXCEPT Statements.
- TurboDB Pilot mit stark verbessertem Komfort bei der Verwaltung von Datenbanken und Kommandos.
- TurboDB Pilot mit visuellem Designer zum Erstellen und Ändern von Datenbank-Tabellen.

Version 1.3

TurboDB Managed 1.3 beinhaltet viele neue Features rund um die Programmierung.

- Benutzerdefinierte Funktionen ermöglichen die Implementierung von eigenen SQL Funktionen entweder in SQL oder als .NET Assembly.
- Mit Stored Procedures können komplexe Befehlssequenzen mit einem einzigen Aufruf abgearbeitet werden. Sie werden entweder in TurboSQL oder in einer beliebigen .NET Sprache implementiert.
- User-defined Aggregates erlauben die Definition neuer Aggregatsfunktionen in einer beliebigen -NET Sprache zur Verwendung in SQL Statements.

Lesen Sie das Kapitel "Programmiersprache" für weitere Informationen.

1.3.1.4 Upgrade auf TurboDB Managed v2

TurboDB Managed 2.0 ist voll-kompatibel zu Version 1.x.

Neue reservierte Schlüsselwörter

Es wurden einige neue Schlüsselwörter eingeführt. Wurde eines davon als Tabellen- oder Spalten-Name verwendet, muss dieser in Statements nun in eckigen Klammern [...] oder Anführungszeichen "..." geschrieben werden: BY, CORRESPONDING, ENCRYPTION, EXCEPT, INTERSECT, UNION, CONTAINS, FULLTEXTINDEX.

1.3.2 TurboDB Engine Konzepte

Dieses Kapitel beschreibt die grundlegenden Konzepte der TurboDB Database Engines.

Überblick behandelt Leistungsmerkmale, Namenskonventionen und Datentypen.

Datenbanken erklärt den Unterschied zwischen Single-File und Directory Datenbanken.

Indexe behandelt die verschiedenen von TurboDB unterstützten Index-Arten.

Automatic Data Linking präsentiert das TurboDB Konzept für ein schnelleres und weniger Fehler-anfälliges Datenmodell.

Mehrbenutzerzugriff und Sperren beschreibt wie TurboDB Multi-User Access implementiert.

Optimierung bietet eine Liste von Punkten, die Sie prüfen können falls Ihre Anwendung nicht schnell genug ist.

Fehlerbehandlung beschreibt welche Fehler bei der Arbeit mit der Datenbank auftreten können und wie diese zu behandeln sind.

Verschiedenes behandelt die physikalische Datenbanktabellen, Datensicherheit und Lokalisierung.

1.3.2.1 Überblick

Leistungsmerkmale

Tabellen- und Spaltennamen

Datentypen für Tabellenspalten

1.3.2.1.1 Kompatibilität

Es gibt zwei Implementierungen der TurboDB Datenbank Engine, die Win32 Engine und die .NET Engine, letztere auch als Managed Engine bezeichnet. Die beiden Datenbank Engines können mit identischen Datenbank-Dateien arbeiten, solange die Einschränkungen der beiden Engines berücksichtigt werden. Das sind die zu beachtenden Regeln, wenn Sie Datenbank-Dateien sowohl mit TurboDB Managed 2.x als auch mit TurboDB für Win32 5.x bearbeiten wollen.

- Verwenden Sie eine Single-File- statt einer Directory-Datenbank
- Verwenden Sie maximal 40 Zeichen für Tabellen- und Spaltennamen.
- Verwenden Sie nicht *Blowfish* Verschlüsselung, sondern *FastEncrypt* oder *Rijndael*. Das Passwort für alle Tabellen muss identisch sein, da TurboDB Managed nur ein Passwort für eine Datenbank unterstützt.
- TurboDB für Win32 ignoriert benutzerdefinierte Funktionen und Stored Procedures. Sie können weder in berechneten Indexen noch in Queries benutzt werden.
- Verwenden Sie keine Sprachtreiber.

- Da sich der Sperrmechanismus unterscheidet können TurboDB Managed und TurboDB für Win32 nicht dieselbe Datenbank gleichzeitig öffnen. Ein abwechselnder Zugriff ist möglich.

1.3.2.1.2 Systemvoraussetzungen

TurboDB für .NET

.NET Framework ab Version 2.0

TurboDB für Win

Betriebssystem: Windows 32 Bit ab Windows 2000 und Windows 64 Bit ab Windows Vista - alle von Microsoft unterstützten Editionen ohne Windows Phone/Windows RT.

Es gibt 32 Bit und 64 Bit Versionen von TurboDB. TurboDB läuft auf den 64-Bit-Versionen der Betriebssysteme auch im 32-Bit Modus ohne Probleme.

1.3.2.1.3 Leistungsmerkmale

Einige technische Daten zu Tabellen Level 4 und höher:

Maximale Anzahl Zeilen pro Tabelle	2 G
Maximale Tabellengröße	4 EB (ein Exabyte ist 1G mal 1GB)
Maximale Anzahl Spalten pro Tabelle	1.000
Maximale Größe einer Zeile	32 KB
Maximale Anzahl Benutzer-definierter Indexe pro Tabelle	48 (ab Tabellen-Level 4) oder 13 (bis Tabellen-Level 3)
Maximale Anzahl Ebenen in hierarchischen Indexen	255
Maximale Größe der Index-Information	32 KB
Maximale Anzahl Tabellen pro Datenbank	255
Maximale Länge einer String-Spalte	32.767
Maximale Anzahl von Link-Feldern pro Tabelle	9
Maximale Gesamtgröße aller Blobs einer Tabelle	1 TB (bei Blockgröße 512 B) bis 64 TB (bei Blockgröße 32 KB)

1.3.2.1.4 Tabellen- und Spaltenamen

Bezeichner

Bezeichner bestehen aus einem echten Buchstaben gefolgt von alphanummerischen Zeichen, dem Unterstrich_ und dem Bindestrich. Sie dürfen maximal 40 Zeichen lang sein. Deutsche Umlaute sind erlaubt.

Gültige Bezeichner sind:

```
Name
Alter
Straße
Durchschitts_Dauer
Tag_der_Geburt
Adresse8
Lösung
```

Spalten- und Tabellen-Namen

Spalten- und Tabellen-Namen können alle ANSI-Zeichen beinhalten außer Steuercodes, eckige Klammern [] und doppelte Anführungszeichen. Sie sind auch maximal 40 Zeichen lang und müssen mit einem echten Buchstaben beginnen. Entspricht der Name den Regeln für Bezeichner, kann er in allen ausdrücken und Statements ganz normal benutzt werden. Falls nicht, muss er entweder in doppelten Anführungszeichen oder in eckige Klammern gesetzt werden.

Diese Beispiele sind **ungültige** Bezeichner, die trotzdem als Spalten-Namen verwendet werden können, indem sie in doppelte Anführungszeichen oder eckige Klammern gesetzt werden:

```
Anzahl Einträge(Leerzeichen nicht erlaubt)
3645 (das erste Zeichen muss ein Buchstabe sein)
```

1.3.2.1.5 Datentypen für Tabellenspalten

TurboDB bietet die folgenden Datentypen für Tabellen-Spalten:

String

Ein String Feld beinhaltet alphanummerische Zeichen bis zu einer bestimmten Länge. Die maximale Länge ist 32.767 Zeichen (255 Zeichen bis Tabellen-Level 3). Ein String Feld stellt ein Byte pro Zeichen zur Verfügung plus ein oder zwei Byte für die Länge der Zeichenkette, plus ein Byte falls die Spalte über einen Null-Indikator verfügt.

WideString

Bis zu 32767 Unicode Zeichen (255 bis Tabellen-Level 3). Da ein Unicode Zeichen 2 Bytes benötigt, ist die resultierende Feldgröße zwei mal die Anzahl der Zeichen, plus 2 Byte für die Länge, plus ein Byte falls das Feld über einen Null-Indikator verfügt.

Byte

Zahlen von 0 bis 255. Byte Felder können optional einen Null Indikator haben. Die Größe ist ein oder zwei Bytes, abhängig davon ob ein Null Indikator verwendet wird oder nicht.

SmallInt

Zahlen von -32767 bis +32768. SmallInt Felder können einen optionalen Null-Indikator haben. Die Größe ist zwei oder drei Bytes.

Integer

Zahlen von -2147483648 bis +2147483647. Integer Felder können einen optionalen Null Indikator haben. Es werden vier oder fünf Bytes in der Datenbanktabelle benötigt.

LargeInt

Zahlen von –2^63 bis +2^63 – 1 mit einem optionalen Null Indikator. Ein Int64 Feld verwendet acht oder neun Byte in der Tabelle.

Float

Speichert eine 8 Byte Fließkommazahl, z.B. von 5.0e-324 bis 1.7 x 10e308. Ein optionaler Null Indikator ist möglich. Die Größe ist acht oder neun Bytes.

Zeit

Für Tabellen-Level 4: Werte von 00:00:00.000 bis 23:59:59.999. Die Genauigkeit wird bei der Erstellung der Spalte festgelegt und beträgt entweder Minuten, Sekunden oder Millisekunden. Für Tabellen-Level bis 3 ist die Genauigkeit immer Minuten. Ein optionaler Null-Indikator ist möglich. Die Größe ist, abhängig von der Genauigkeit und dem Null-Indikator, zwei bis fünf Bytes.

Datum

Werte von 1.1.0000 bis 31.12.9999. Das Feld benötigt vier Bytes. Intern werden Datumsfelder als gepacktes Bitfeld dargestellt.

DateTime

Werte von 1.1.0000 00:00 bis to 31.12.9999 23:59. Es werden acht Byte benötigt.

Boolean

Speichert die logischen Werte True oder False. Ein optionaler Null Indikator ist möglich. Die Größe ist ein oder zwei Bytes.

Auswahl

Ein Auswahl Typ definiert eine Liste von lesbaren, aussagekräftigen Ausdrücken und speichert den Wert in einem einzigen Byte in der Tabelle ab. Es ist dasselbe Prinzip wie beim Aufzählungstyp von Delphi, läßt sich aber direkt auf Datenbank Tabellen anwenden.
Sie können zum Beispiel ein Feld mit Namen Geschlecht haben und die Aufzählungswerte könnten mit weiblich, männlich und unbekannt definiert sein. Auf diese Weise haben Sie automatisch einen verständlichen und sicheren Weg die Werte des Feldes anzuzeigen und zu editieren und gleichzeitig werden die Werte sehr effektiv abgespeichert. Aufzählungswerte können selbstverständlich auch in Filter- und Suchausdrücken verwendet werden: 'Geschlecht = männlich'.
Ein Aufzählungswert darf bis zu 20 Zeichen lang sein, bis zu 15 Aufzählungswerte sind möglich und die Gesamtlänge aller Aufzählungswerte zusammen inklusive Trennzeichen dazwischen darf nicht länger als 255 Zeichen sein.

Memo

Lange Zeichenketten mit variabler Länge (bis 1G). Memos werden in einer zusätzlichen Datei gespeichert, die denselben Namen hat wie die Datenbanktabelle, aber mit der Erweiterung mmo/tdbm.

Wid**e**Memo

Unicode String variabler Länge (bis 1G). Wide Memos werden in BLOBs gespeichert (Dateierweiterung blb/tdbb).

Blob

Bilder und andere binäre Daten variabler Länge. Blobs werden in einer zusätzlichen Datei gespeichert, die denselben Namen hat wie die Datenbanktabelle, aber mit der Erweiterung blb/tdbb.

Link

Link-Felder sind Zeiger auf Datensätze in einer anderen Tabelle. (1:n Beziehung). Linkfelder werden in Die "Automatic Data Link Technologie" beschrieben. Ein Linkfeld speichert den Wert des AutoInc-Feldes des Datesatzes auf den das Link-Feld verweist. Seine Größe ist vier Bytes.

Relation

Relationsfelder enthalten virtuell eine Liste von Zeigern, auf diese Weise ermöglichen sie n:m Beziehungen. Relationen werden in Die "Automatic Data Link Technologie" beschrieben. Physikalisch werden Relationfelder durch eine zusätzliche Relationstabelle realisiert, die über zwei Linkfelder verfügt um die n:m Beziehung abzubilden. Daher benötigt ein Relationsfeld selbst 0 Bytes.
Nicht verfügbar in TurboDB Managed

AutoInc

Automatisch generierte, eindeutige 32-Bit Zahl. AutoInc Felder werden von der Datenbank Engine verwaltet und können daher nicht editiert werden. Die "Automatic Data Link Technologie" verwendet AutoInc Felder als Primärindex um Referenzen auf Datensätze zu speichern.

GUID

128 Bit Zahl, die von MS COM und ActiveX Technologien verwendet wird. GUID steht für Globally Unique Identifier. GUIDs werden für gewöhnlich durch eine Betriebssystem-Funktion berechnet, die sicherstellt, dass dieser Wert weltweit einzigartig ist.

Hinweis

Bei den Datentypen AutoInc, Link und Relation werden automatisch ein oder mehrere Indexe erstellt. Diese (System)-Indexe tragen je nach Tabellenlevel den Namen der Tabelle oder beginnen mit dem Prefix 'sys_'. Ein Ändern oder Löschen dieser Indexe ist nicht möglich.

Siehe auch

TurboSQL Column Types

1.3.2.1.6 Kollationen

Kollationen definieren auf welche Weise Zeichenketten sortiert und miteinander verglichen werden. TurboDB verwendet für Kollationen ein ähnliches Schema zur Namensgebung wie Microsoft SQL Server. Der Name einer Kollation besteht aus dem Namen eines Windows Gebietschemas plus zwei oder vier zusätzlichen Buchstaben, die anzeigen, wie Groß-Kleinschreibung und diakritische Zeichen behandelt werden sollen.

Die Bedeutung dieser vier Zeichen ist:

- *AS*: Diakritische Zeichen (Akzente) unterscheiden
- *AI*: Diakritische Zeichen (Akzente) nicht unterscheiden
- *CS*: Klein- und Großbuchstaben unterscheiden
- *CI*: Klein- und Großbuchstaben nicht unterscheiden

Falls beides angegeben wird, muss die Spezifikation der Groß-Kleinschreibung vor der Angabe der diakritischen Zeichen stehen. Es ist auch möglich keine Spezifikation anzugeben. In diesem Fall werden diakritische Zeichen unterschieden, Groß-Kleinschreibung dagegen nicht.

Neben den Namen der Windows Gebietsschemas dient die spezielle Kollation *TurboDB* zur Sortierung wie Sie in TurboDB 5 und kleiner angewendet wurde. Demzufolge ist es nicht möglich die *as/ai/cs/ci* Spezifikationen an den Kollations-Namen *TurboDB* anzuhängen. Die Namen der Kollationen selbst sind case insensitiv.

Hier einige Beispiele für gültige Namen von TurboDB Kollationen:

- *German* (Diakritische Zeichen unterscheiden, Groß-Kleinschreibung nicht unterscheiden)
- *GERMAN* (wie oben)
- *English_ai* (Diakritische Zeichen nicht unterscheiden, Groß-Kleinschreibung nicht unterscheiden)
- *Spanish_ci_as* (Diakritische Zeichen unterscheiden, Groß-Kleinschreibung nicht unterscheiden)
- *Russian_cs* (Diakritische Zeichen unterscheiden, Groß-Kleinschreibung unterscheiden)
- *Russian_CS* (wie oben)
- *TurboDB* (Diakritische Zeichen unterscheiden, Groß-Kleinschreibung nicht unterscheiden)

Hier einige Beispiele für **ungültige** Namen::

- *Collation1_ai* (Collation1 ist keine Windows Gebietsschema)
- *Spanish_as_ci* (Falsche Reihenfolge der zusätzlichen Spezifikationen)
- *Spanish_ca* (Ungültige Spezifikation)
- *TurboDB_cs* (Die spezielle Kollation *TurboDB* darf keine zusätzlichen Spezifikationen haben)

Tipp: Verwenden Sie den Tabelleneditor des TurboDB Viewer um eine Liste der auf Ihrem System verfügbaren Kollationen anzuzeigen..

Kollationen können auf mehrere Arten zugewiesen und ausgelesen werden:

- TurboDB Viewer zeigt Kollationen an und ermöglicht die Auswahl, wenn eine Tabelle erzeugt oder restrukturiert wird.
- TurboSQL unterstützt sie Klausel COLLATE für Spaltentypen.
- In der VCL Bibliotek verfügt die Klasse *TTdbFieldDef* über eine Eigenschaft namens *Specification*, in der für String Typen der Name der Kollation steht.
- Die .NET Provider unterstützen Kollationen über das entsprechende ADO.NET Interface.

Kompatibilität

Kollationen werden sowohl von TurboDB für Win32 v6 aös auch von TurboDB für .NET v3 unterstützt. Aussschließlich Tabellen mit Level 6 oder höher erlauben die Definition von Kollationen auf Spaltenebene. Tabellen mit niedrigerer Versionen erlauben lediglich die Definition von Kollationen auf Tabellenebene, wobei der Dreibuchstaben ISO Code für die Sprache verwendet wird. Ältere TurboDB Versionen verwendeten optional Sprachtreiber,die nicht mehr unterstützt werden. Hinweise zur Migration dieser Tabellen finden Sie im Kapitel Upgrade auf TurboDB Win32 v6.

Da String-Vergleiche jetzt zu 100% konsistent in Filtern, SQL, TurboPL und Indexen sind, sind Vergleiche in älteren TurboDB Tabellen jetzt case insensitive. Daher ist der Vergleich *MyStringColumn = 'TestString'* wahr für einen *MyStringColumn*-Wert *'teststring'* in TurboDB 6, wohingegen er falsch war mit TurboDB 5 und früher. Falls case sensitive Vergleiche erwünscht sind, muss eine andere Kollation für die Tabelle oder Spalte definiert werden..

Siehe auch

CREATE TABLE Statement

1.3.2.2 Datenbanken

TurboDB bietet zwei verschiedene Arten von Datenbanken.

Single-File vs. Datenbank-Verzeichnisse

Datenbank-Verzeichnisse

Datenbank-Verzeichnisse sind Verzeichnisse auf der Festplatte, die alle unterschiedlichen TurboDB Datenbank-Objekte in einzelnen Dateien enthalten. Datenbank-Verzeichnisse werden von TurboDB schon immer unterstützt.

Singe-File Datenbank

Eine Singe-File Datenbank ist eine einzelne Datei, die alle Objekte der Datenbank beinhaltet. Ein Datenbank-File hat die Datei-Erweiterung *.tdbd. Single-File Datenbanken werden von TurboDB seit Version 4 unterstützt. Sie haben den Vorteil, der sehr einfachen Weitergabe und können unproblematisch auf der Festplatte kopiert und verschoben werden.

Der Vorteil von Datenbank-Verzeichnissen dagegen ist, dass der Zugriff ein bisschen schneller ist einzelne Tabellen in mehreren Datenbanken verwendet werden können.

Single-File Datenbanken werden über ein virtuelles Dateisystem realisiert, das von dataWeb implementiert wurde. Diese Schicht bildet die Datenbank-Objekte entweder auf die verschiedenen Dateien in einem Verzeichnis oder auf eine einzelne Datenbank-Datei ab. dataWeb bietet dazu ein Werkzeug an, den dataWeb Compound File Explorer, das Datenbank-Dateien öffnen, den Inhalt anzeigen kann und editieren. Das ist ein Weg um Verzeichnis-basierte Datenbanken in eine Single-File Datenbank zu überführen.

Während TurboDB Native beide Datenbank-Typen unterstützt, kann TurboDB Managed ausschließlich mit Single-File Datenbank arbeiten.

Datenbankkataloge

In früheren Versionen waren TurboDB Datenbanken lediglich Sammlungen von Datenbanktabellen, die in separaten Dateien abgespeichert wurden. Während dieser Ansatz den Vorteil hat, dass einzelne Tabellen Bestandteil mehrerer Datenbanken sein können, gibt es auch

einige Nachteile. Einer ist, dass der Name einer Tabelle nicht immer aufgelöst werden kann, da sich die Tabellendatei in einem weit entfernten Verzeichnis befinden kann. Falls die einzelnen Tabellen auf unterschiedliche Weise verschlüsselt sind ist ein weiterer Nachteil, dass verschiedene Passwörter benötigt werden um die Datenbank zu öffnen

Daher führt TurboDB 5 für Win32 Datenbankkataloge ein, die eine Liste der Tabellen und zusätzliche Datenbank-weite Eigenschaften speichern. Kataloge sind sehr hilfreich bei Verwendung von Verzeichnis-Datenbanken. Für Single-File Datenbanken existieren die genannten Nachteile nicht, oder sind weniger problematisch.

TurboDB Managed unterstützt ausschließlich Single-File Datenbanken, verfügt über alle nötigen Informationen und bietet daher keine explizite Katalogfunktionalität.

Hinweis: Datenbanken mit Katalogfunktionalität wurden in früheren Versionen als verwaltete (managed) Datenbanken bezeichnet. Dieser Terminus führt aber zu einem Namenskonflikt mit dem Produkt TurboDB Managed.

1.3.2.2.1 Sessions und Threads

Seit TurboDB Version 4 muss eine Session erzeugt werden, bevor Tabellen und Queries geöffnet werden können. Falls Sie eine Komponenten-Bibliothek verwenden (TurboDB für VCL) werden Sessions im Datenbank- oder Connection-Objekt versteckt verwaltet.

Sie können so vielen Sessions erzeugen wie Sie möchten, sollten aber die Konsequenzen einer Multi-Session Anwendung kennen:

- Cursor der gleichen Tabelle in verschiedenen Sessions werden auf Dateiebene synchronisiert. Das ist wesentlich langsamer als die Synchronisation der Cursor in einer Session, die im Speicher durchgeführt wird.
- Sie können verschiedene Threads für verschiedene Sessions einsetzen, aber nicht verschiedene Threads für eine Session. Aus Gründen der Performanz gibt es keine eingebaute Thread-Synchronisation innerhalb einer Session.

1.3.2.2.2 Tabellen-Levels

Im Zuge des Ausbaus und der Verbesserung von TurboDB wurden im Laufe der Zeit verschiedene Speicherformate entwickelt. Diese Format werden Tabellen-Level genannt und im folgenden beschrieben. Während TurboDB Native alle Level unterstützt, arbeitet TurboDB Managed ausschließlich mit Level 5.

Table Level 6

- Verwendet Dateierweiterungen nach dem Schema tdb? für alle Dateien.
- Benennt Relationstabellen nach den zugehörigen Haupttabellen.
- Unterstützt Windows Kollationen auf Tabellen- und Spaltenebene.
- Bietet Volltextindizierung mit definierbaren Separatoren.
- Verfügt über spezifische, für Textspalten optimierte, Indexstrukturen.
- Unterstützt die Verschlüsselung der Indexe.

Table Level 5

- Ist kompatibel mit TurboDB Managed.
- Unterstützt Gültigkeitsbedingungen, Vorgabewerte, berechnete Spalten und Indexe in Standard SQL.
- Unterstützt gleichnamige Tabellen in unterschiedlichen Datenbanken im selben Verzeichnis.
- Unterstützt die Verschlüsselung des Tabellenschemas.
- Ist vorbereitet für die Unterstützung von Zeichensätzen.
- Ist vorbereitet für Unicode Tabellen- und Spaltennamen.

Table Level 4

- Unterstützt Primärschlüssel und eindeutige Schlüssel.
- Unterstützt Zeit-Spalten mit einer Genauigkeit bis zu Millisekunden.
- Unterstützt zusätzliche Verschlüsselungs-Algorithmen (strong encryption).
- Unterstützt Gültigkeitsbedingungen und Fremdschlüssel.
- Passt die Sortierung an den SQL Standard an, so dass Null-Werte kleiner als alle anderen sind.
- Erhöht die Zahl der Indexe, die für eine Tabelle möglich sind.
- Unterstützt sichere Verschlüsselung.
- Ermöglicht gewartete Volltext-Indexe.

Table Level 3

- Neu sind Unicode Strings, DateTime and GUID Spalten.

Table Level 2

- Führt 32 Bit Integer und ANSI Strings ein.

Table Level 1

- Das erste Datei-Format. Kompatibel mit TurboDB für DOS.

1.3.2.3 Indexe

Indexe werden zusätzlich zu den eigentlichen Tabellen verwaltet, um schnelles Suchen und Sotieren zu ermöglichen. TurboDB Indexdefinitionen bestehen entweder aus einer Liste von Feldnamen oder einem Ausdruck. Falls ein Index als eindeutig definiert ist, werden Datensätze die diese Eindeutigkeit verletzen nicht akzeptiert. Eine weitere Form eines Index ist der Volltextindex.

Indexe basierend auf einer Liste von Feldern

Diese Indexe werden in der Reihenfolge des ersten Feldes in der Feldliste sortiert. Wenn zwei Datensätze den gleichen Wert für das erste Feld haben, werden sie nach dem zweiten Feld der Feldliste sortiert und so weiter. Es kann bis 10 Felder in der Indexfeldliste geben. Jedes Feld kann in auf- oder absteigender Reihenfolge in die Indexbeschreibung eingehen.

Indexe basierend auf einem Ausdruck

Diese Indexe werden nach dem Wert eines beliebigen Ausdrucks sortiert, der bis 40 Zeichen lang sein kann. Ist der Ausdruck vom Type String, wird der Index wie die Werte einer alphanummerischen Spalte sortiert. Wenn der Ausdruck nummerischer Art ist, wird der Index entsprechend der normalem nummerischen Reihenfolge sortiert.

Volltext-Indexe

Ein Volltext-Index erlaubt es einem Anwender in jedem Feld der Tabelle nach einem Schlüsselwort oder einer Reihe von Schlüsselwörtern zu suchen. Volltext-Index benötigen eine separate Tabelle, die Wörterbuch-Tabelle, welche die indizierten Wörter enthält. Volltext-Indexe sind für Tabellen ab Level 4 anders implementiert als für die niedrigeren Level. Ab Level 4 gibt es nur ein Speicherobjekt, das die Verbindung zwischen Wörterbuch-Tabelle und indizierter Tabelle herstellt. Die Dateierweiterung dieses Objekt ist fti oder tdbf. In den älteren Tabellen-Formaten wird Verbindung über Relations-Felder hergestellt, was eine zusätzliche Datenbank-Tabelle (Extension rel) und zwei weitere Indexe (in1 und in2) nötig macht.

Indexe können mit verschiedenen TurboDB Tools zur Entwurfszeit erzeugt, geändert und gelöscht werden. Zur Laufzeit kann TurboSQL oder Methoden der VCL Komponenten Bibliothek verwendet werden um Volltext-Index zu erstellen, zu erneuern oder zu löschen.

System-Indexe

Bei den Datentypen AutoInc, Link und Relation werden automatisch ein oder mehrere Indexe erstellt. Diese (System)-Indexe tragen je nach Tabellenlevel den Namen der Tabelle oder

beginnen mit dem Prefix 'sys_'. Ein Ändern oder Löschen dieser Indexe ist nicht möglich.

1.3.2.4 Automatic Data Link

In den allermeisten Fällen sind Tabellenverknüpfungen für alle Abfragen gleich. Z.B. gehören einzelne Posten immer zu einer Rechnung, Autoren sind immer mit den Büchern verknüpft, die sie geschrieben haben u.s.w. Daher ermöglicht es TurboDB die verschiedenen Arten von Verknüpfungen von einer Tabelle zu anderen Tabellen in der Tabelle selbst zu definieren.

Stellen Sie sich vor, Sie haben eine Rechnungstabelle, die das Rechnungsdatum, die Kundennummer, die Rechnungsnummer und andere rechnungsbezogene Informationen enthält. Die einzelnen Rechnugsposten werden in einer anderen Tabelle eingetragen, die Felder für die Artikelnummer, den Preis, den Gesamtbetrag und andere enthält. Wie verknüpfen Sie den Einzelposten mit der Rechnung deren Bestandteil der Posten ist? Der traditionelle Weg ist es in der Postentabelle ein zusätzliches Feld zu haben, das die Nummer der Rechnung aufnimmt, zu der der Posten gehört. Jede Abfrage, die die Rechnungs-Posten Beziehung berücksichtigt, muss die folgende Bedingung formulieren: ...where "POSTEN.RechnungsNummer" = "RECHNUG.RechnungsNummer"...

Was ist das?

Sie können diesen traditionellen Weg auch mit TurboDB gehen, die bevorzugte Lösung ist ein bisschen anders. Anstatt ein RechnungsNummer Feld in der POSTEN Tabelle zu haben würden Sie lieber einen Zeiger auf die Rechnungstabelle verwenden. Dieser Zeiger wird als Linkfeld bezeichnet. Da TurboDB standardmäßig eine (eindeutige) RecordId in jede neue Tabelle einfügt, speichert das Linkfeld der Postentabelle nur die RecordId der Rechnung zu der der Posten gehört. Da die Definition des Linkfeldes die Information beinhaltet, dass der Inhalt dieser Spalte auf einen Eintrag in der Rechnungstabelle zeigt, weiß die Datenbank über diese Beziehung bescheid und wird dies standardmäßig bei jeder Abfrage berücksichtigen.

Diese Art Tabellen zu verknüpfen hat einige große Vorteile:

Abfragen über verknüpfte Tabellen sind sehr einfach, da das System "weiß" wie die Tabellen miteinander zu verknüpfen sind. Die Abfragen können viel schneller sein, weil eine RecordId nur eine Zahl ist, während Sekundärindexe oftmals wesentlich komplexer sind. Das Ändern von Indexen, Spaltennamen oder Spaltentypen beläßt die Verknüpfung unberührt. Über eine spezielle Link Notation ist es sehr einfach auf den Masterdatensatz zuzugreifen.

Man kann diese Strategie als den objektorientierten Weg betrachten mit Datenbanktabellen zu arbeiten. Sie ist nicht streng konform mit dem relationalem Paradigma, bringt aber das Feeling von Zeigern und Referenzen ins Spiel. Der Rechnungsposten "weiß" zu welcher Rechnung er gehört. Diese Verbindung liegt in der Natur der Sache und wird sich sicher nicht sehr oft ändern.

Ein weiterer Vorteil ist, dass Link- und Relationsfelder dem Anwender nicht nur die rein technischen AutoInc-Werte anzeigen können. Jeder AutoInc-Spalte kann eine Anzeigeinformation bestehend aus Werten anderer Spalten zugeordnet werden. Diese wird von Link und Relationsfeldern anstelle der nummerischen Werte angezeigt. Hier ein Beispiel:

CREATE TABLE DEPARTMENT (Name CHAR(20), Id AUTOINC(Name)

CREATE TABLE EMPLOYEE (LastName CHAR(40), Department LINK(DEPARTMENT))

Die Abfrage

SELECT * FROM EMPLOYEE

liefert eine Liste von Nachnamen und Abteilung-Namen, das der Name der Abteilung als Anzeigeinformation für die AutoInc-Spalte *Id* definiert ist.

Wie funktioniert es?

Da Linkfelder auf so einfache Weise 1:n Beziehungen (eine Rechnung hat mehrere Posten) einführen, wirft dieses objektorientierte Konzept die Frage nach m:n Beziehungen als eine Liste von Zeigern auf, die von einer Tabelle auf eine andere zeigen. TurboDB Relationsfelder sind die

Antwort auf diese Frage. Eine Tabelle, die ein Relaionsfeld enthält verknüpft einen Datensatz mit mehreren Datensätzen der anderen Tabelle und umgekehrt. Um zu dem Beispiel mit Autoren und Büchern zurückzukehren, würde das Einfügen eines Relationsfeldes in die Büchertabelle die Tatsache berücksichtigen, dass ein Buch von mehreren Autoren geschrieben und ein Autor durchaus an mehreren Büchern beteiligt sein kann.

Wie Sie sicher schon annehmen sind Relationsfelder nicht so einfach zu implementieren wie Linkfelder. M:n Beziehungen werden durch eine zusätzliche Zwischen-Tabelle realisiert, die einen Datensatz für jede Verknüpfung zwischen den beiden Tabellen erhält. Das genau ist es was TurboDB macht, wenn Sie ein Relationsfeld in die Buchtabelle einbauen das auf die Autorentabelle zeigt. TurboDB wird eine versteckte Zwischen-Tabelle erzeugen, mit einem Linkfeld auf die Autorentabelle und einem Linkfeld auf die Büchertabelle. Das ist es auch was Sie tun müssten wenn Sie auf traditionelle Weise arbeiten würden. Aber mit TurboDB wird die Zwischen-Tabelle automatisch und transparent erzeugt und gepflegt.

Kompatibilität

Dieses Feature wird nur zum Teil in TurboDB Managed unterstützt. In TurboDB Managed gibt es momentan Link-Spalten, aber keine Relations-Spalten.

1.3.2.4.1 Mit Link- und Relationsfelder arbeiten

Um von der Automatic Link Technologie zu profitieren, sollten Sie darüber nachdenken in jeder Tabelle, die Sie neu anlegen, mit Link- oder Relationsfeldern zu arbeiten. Sie werden es sehr bald als völlig natürlich empfinden die Verknüpfungsinformation in der Tabelle zu speichern. Schließlich machen Sie ja dasselbe mit Ihren Delphi, C++ oder Java Klassen, oder?

Hnizufügen von Link- und Relations-Spalten

Wenn Sie mit Link- oder Relations-Felder arbeiten möchten, um eine 1-n oder n-m Beziehung zwischen Tabellen herzustellen, müssen Sie entscheiden welche der Tabellen die Quelle und welcher das Ziel der Beziehung ist. Erstere wird als Kind-Tabelle, zweitere als Eltern-Tabelle bezeichnet. Es verhält sich wie mit der referenzierenden und referenzierten Tabelle, wenn Sie an die Arbeit mit traditionellen Fremdschlüsseln denken.

Die Eltern-Tabelle muss über eine AutoInc-Spalte verfügen,. die als Primärschlüssel für die Verknüpfung dient. Die Kind-Tabelle muss eine Link- oder Relations-Spalte beinhalten, welche die Verknüpfung etabliert. Eine Link-Spalte kann genau einen Zeiger auf die Eltern-Tabelle speichern. Der Zeiger wird als AutoInc-Wert der Eltern-Tabelle angezeigt oder als den Werten der Tabellenspalten der Anzeigeinformation, falls diese für die AutoInc-Spalte der Eltern-Tabelle definiert ist. Relationsfelder können mehrere Zeiger auf die Eltern-Tabelle speichern, die als Liste der AutoInc-Werte oder als Liste der Anzeigeinformationen angezeigt werden.

Link- und Relationsfelder beim direkten Tabellen-Zugriff

(Direkter Tabellenzugriff ist mit der VCL/CLX-Komponenten-Bibliothek möglich, nicht aber mit ADO.NET.)

Wenn Sie Ihre Links und Relationen erst einmal definiert haben, werden diese bei jeder Abfrage von der Datenbank berücksichtigt. Sogar wenn Sie überhaupt kein Suchkriterium angeben, werden wirklich nur verknüpfte Detail-Datensätze zum aktuellen Master-Datensatz angezeigt. Für den seltenen Fall, dass Sie dieses Verhalten nicht möchten, können Sie jederzeit einen anderen Equate Join angeben, der den Standard überschreibt.

Link- und Relationsfelder in TurboSQL

In TurboSQL-Abfragen werden die durch Link- und Relationsfelder definierten Verknüpfungen nicht automatisch hergestellt. Mit einem einfachen JOIN erhält man allerdings den Bezug:

```
SELECT * FROM Master JOIN Detail ON Detail.LinkField = Master.RecordId
```

Zum Eintragen neuer Datensätze in den Verbund, kann man die Funktion CurrentRecordId verwenden:

```
INSERT INTO Master VALUES(...); INSERT INTO Detail VALUES(...,
CurrentRecordId(Master), ...)
```

Mit diesem zusammengesetzten Statement wird zuerst ein Datensatz in die Master-Tabelle

eingetragen und dann sofort ein weiterer Datensatz in die Detail-Tabelle, wobei als Wert für das Link-Feld die letzte RecordId der Master-Tabelle benutzt wird. Dadurch ist der Detail-Datensatz mit dem Master-Datensatz verknüpft.

Komaptibilität

Dieses Feature wird nur zum Teil in TurboDB Managed unterstützt. In TurboDB Managed gibt es momentan Link-Spalten, aber keine Relations-Spalten.

1.3.2.5 Mehrbenutzerzugriff und Sperren

Standardmäßig öffnet TurboDB Tabellen im Shared Mode. Es können also mehreren Anwendungen gleichzeitig auf eine Datenbanktabelle zugreifen. Um inkonsistente Änderungen der Tabelle zu vermeiden, gibt es einen transparenten Sperrmechanismus, der Datensätze für die Dauer des Editiervorganges durch eine Anwendung sperrt.

Einige Operationen benötigen allerdings exklusiven Zugriff auf eine Datenbanktabelle und werden daher zurückgewiesen falls eine weitere Anwendung die Tabelle benutzt. Diese Operationen sind:

- Ändern der Tabellenstruktur (AlterTable)
- Löschen einer Tabelle (DeleteTable)
- Löschen eines Index

Lock Files

Da TurboDB eine dateibasierte Datenbank Engine ist, werden Tabellensperren über eigene Verwaltungsdateien verwaltet. Diese Lock Dateien haben denselben Namen wie die entsprechende Datenbanktabelle, aber mit der Dateierweiterung "net". Die Lock Datei beinhaltet eine Liste aller Anwendungen, die auf die Datenbanktabelle zugreifen und verwaltet die unterschiedlichen Sperren auf diese Tabelle.

Wenn eine Anwendung in eine Datenbanktabelle schreiben will, muss sie erst in der Lock Datei nachsehen, ob es erlaubt ist. Falls ja, registriert sich die Anwendung in der Lock Datei als schreibend, führt die Aktion aus und meldet sich wieder ab. Die Lock Datei wird von der ersten Anwendung erzeugt, die auf die Tabelle zugreift und wieder gelöscht, wenn die letzte Anwendung den Zugriff beendet. Daher verfügt eine Tabelle über keine Lock Datei, solange sie nicht geöffnet wird.

Bemerkung: Falls eine Anwendung abstürzt oder beendet wird, während eine Sperre auf die Tabelle oder einen Datensatz eingerichtet ist, kann die Lock Datei nicht gelöscht werden und die Sperre bleibt bestehen. Das kann gerade beim Debuggen einer TurboDB Anwendung des öfteren passieren, da ein Reset während eine Sperre eingerichtet ist genau diesen Effekt hat. Seit Version 4 hat TurboDB einen einzigartigen Mechanismus, der solche toten Sperren erkennt und automatisch aufhebt.

1.3.2.5.1 Tabellensperren

TurboDB Engine arbeitet mit zwei verschiedenen Arten von Tabellensperren:

Eine **Lesesperre** (shared lock) hält andere Anwendungen vom Schreiben in die Tabelle ab. Lesesperren garantieren eine konsistente Datenbanktabelle während einer Abfolge von Leseoperationen, z.B. beim Erstellen eines Index. Es können mehrere Anwendungen gleichzeitig eine Lesesperre für eine Tabelle setzen.

Eine **Schreibsperre** (exclusive lock) verhindert jeden Zugriff einer anderen Anwendung auf die Tabelle. Schreibsperren werden für verschiedene Schreiboperationen benötigt, z.B. das Ändern der Tabellenstruktur. Nur jeweils eine Anwendung kann eine Schreibsperre für eine Tabelle definieren.

Eine TurboDB Tabelle kann auch exklusiv geöffnet werden. In diesem Modus wird auch die zugrundeliegende Datenbankdatei exklusiv geöffnet, was jeder anderen Anwendung unmöglich macht auf die Datei zuzugreifen. Eine Tabelle exklusiv zu öffnen ist einer Schreibsperre ähnlich, mit einigen wesentlichen Unterschieden:

- Schreibsperren werden von TurboDB verwaltet, der exklusive Modus dagegen vom Betriebssystem.
- Schreibsperren können auf eine bereits geöffnete Tabelle angewendet werden. Der exklusive Modus kann nur vor dem Öffnen der Tabelle gesetzt werden.

Anmerkung

Bei der Arbeit mit den TurboDB VCL Komponenten werden Sie bemerken, dass Lesesperren als write locks und Schreibsperren als total locks bezeichnet werden. Der Grund ist, dass die BDE die Sperrentypen durcheinander bringt und wir versuchen halbwegs kompatibel zu sein.

1.3.2.5.2 Satzsperren

TurboDB sperrt automatisch den Datensatz, wenn ein Anwender mit dem Editieren beginnt. Das verhindert, dass zwei Anwender denselben Datensatz gleichzeitig bearbeiten. Die Sperre wird aufgehoben wenn der Anwender die Änderung bestätigt (Post) oder Rückgängig macht (Cancel) oder wenn er die Tabelle schließt.

Wenn ein Anwender einen gesperrten Datensatz editieren möchte, wird ein Fehler zurückgegeben und der Anwender kann den Datensatz nicht ändern. Die meisten Komponenten-Bibliotheken (z.B VCL/CLX) werfen eine Exception falls der Fehler auftritt.

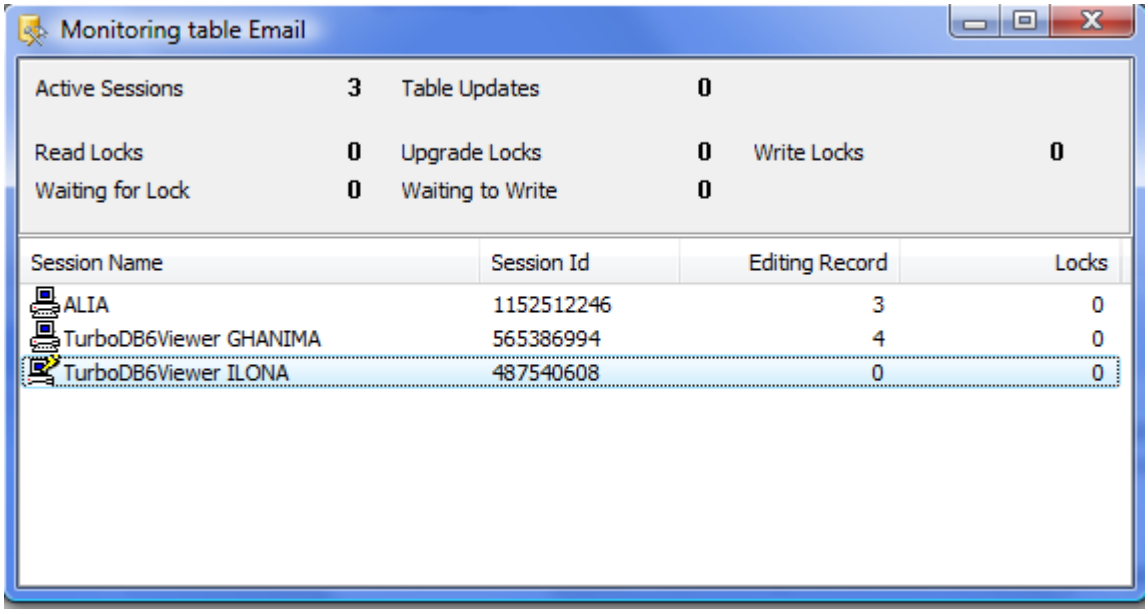
In TurboDB können Satzsperren neben Tabellensperren auftreten. Es ist möglich, dass ein Anwender beginnt einen Datensatz zu editieren und ein zweiter führt ein Update Statement auf die Tabelle aus, bevor der erste Anwender seine Änderung geschrieben hat. Das geht gut, solange das Update nicht den gesperrten Datensatz tangiert. Falls es das tut, wird das Update scheitern. In beiden Fällen kann der editierende Anwender die durchgeführten Änderungen schreiben.

Für einige Anwendungen mit sehr vielen Usern in einem losen Netzwerk (z.B. Web-Anwendungen), ist es keine gute Idee Datensatzsperren aufrechtzuerhalten während der Anwender den Satz editiert. In diesem Fall sollte ein unverbundenes Datenzugriffsmodell bevorzugt werden.

1.3.2.5.3 Anzeige der Tabellen-Nutzung

Da Sperren einen Einfluß auf die Performanz haben können und in manchen Fällen die korrekte Ausführung eines Programms verhindern (Deadlock), ist es manchmal hilfreich sich einen Überblick über die verschiedenen Anwendungen und User zu verschaffen, die auf eine Tabelle zugreifen. Dazu kann man einen Blick in die *.net/*.tdbl Datei der Tabelle werfen, die alle nötigen Informationen beinhaltet.

TurboDB 6 Viewer hat dazu den Menü-Befehl *Table/Access Monitor*, der den Netzwerkmonitor anzeigt.



Eine andere Mehode um den Inhalt der Net-Datei zu prüfen bietet das Kommandozeilen-Tool TdbLocks.exe, das für Windows im Download-Bereich der dataWeb Website bereitsteht.

Wenn Sie die Tabellen-Nutzung in Ihrer Anwendung überprüfen möchten, können Sie das, wenn Sie mit den VCL Komponenten von Delphi arbeiten, über die Methode TTdbTable.GetUsage erreichen. (ADO.NET arbeitet mit einem unverbundenen Datenmodel und kümmert sich daher recht wenig um Sessions die eine Tabelle verwenden.)

Egal auf welche Weise Sie einen Blick in die Net-Datei riskieren, die Information, die Sie vorfinden werden ist immer dieselbe:

Zuerst kommt die Tabellen-bezogene Information:

Active Sessions	Anzahl der momentanen Verbindungen, die diese Tabelle verwenden
Table Updates	Anzahl der Number Änderungen seitdem die Tabelle von der ersten Verbindung geöffnet wurde
Read Locks	Anzahl der Sitzungen, die eine Lese-Sperre auf die Tabelle eingerichtet haben.
Upgrade Locks	Anzahl der Sitzungen, die eine Aktualisierungs-Sperre auf die Tabelle eingerichtet haben.
Write Locks	Anzahl der Sitzungen, die eine Schreib-Sperre auf die Tabelle eingerichtet haben.
Waiting for Lock	Anzahl der Sitzungen, die momentan darauf warten eine Sperre einzurichten
Waiting to Write	Anzahl der Sitzungen, die momentan eine Lese-Sperre halten und darauf warten eine Schreib-Sperre einzurichten

Darauf folgt die Session-bezogene Information:

Session Name	Lesbarer Name einer Sitzung zur leichteren Identifikation. Der Name kann per Programmierung der Datenbank-Komponente festgelegt werden. Ist dies nicht der Fall, wird automatisch ein Name nach dem Schema ConnXXX erzeugt, wobei XXX eine Zufallzahl ist.
Session Id	Automatisch erzeugte, eindeutige Identifikation der Session
Editing Record	Der Datensatz, den die Sitzung momentan editiert (falls dies der Fall ist)
Locks	Korrespondiert mit Write Locks der Tabellen-bezogenen Information. Kann

also die Werte -1, 0 oder 1 haben.

1.3.2.6 Transaktionen

TurboDB unterstützt Transaktionen basierend auf einem zusätzlichen Speicherobjekt pro Tabelle, dem Redo-Log. Nach dem Start einer Transaktion werden alle folgenden Änderungen an den Datenbank-Tabellen in die Redo-Logs eingetragen. Ist die Transaktion mit Commit abgeschlossen, werden die Redo-Logs einfach gelöscht. Bei einem Rollback wird die im Redo-Log vorhandene Information verwendet um die Änderungen rückgängig zu machen. Das bedeutet, TurboDB folgt einem optimistischen Transaktionsschema: Das Committen einer Transaktion ist sehr schnell, während ein Rollback wesentlich aufwändiger ist.

Die während einer Transaktion geänderten Tabellen sind für andere Anwendungen gesperrt, bis die Transaktion abgeschlossen ist. Aus Gründen der Geschwindigkeit werden Tabellen, die während einer Transaktion gelesen werden, nicht gesperrt. Der TurboDB Transaktions-Level ist Read Commited.

Da TurboDB Anwendungen auf Dateiebene miteinander kommunizieren (d.h. ohne Datenbankserver), ist das Handling von Anwendungen, die während einer Transaktion sterben schwierig. TurboDB Engine erkennt, dass eine andere Anwendung gestorben ist und führt ein Rollback aus. Da in diesem Fall eine andere Anwendung die vom gestorbenen Client durchgeführten Änderungen rückgängig macht, nennen wir diesen Mechanismus *Hijacking*.

1.3.2.7 Optimierung

Wenn Sie das Gefühl haben, dass Ihre TurboDB Anwendung langsamer ist als sie sein sollte, kann es dafür viele Gründe geben. Prüfen Sie die folgenden Punkte und befolgen Sie die entsprechenden Ratschläge.

Abfragen gegen eine große Tabelle oder gegen mehrere Tabellen dauern sehr lange

Prinzipiell gibt es zwei Wege eine Abfrage zu beschleunigen: Die nötigtigen Indexe erstellen und/oder das Statement optimieren.

Filtern mit einer Tabellen Komponente (VCL) dauert sehr lange

Einen Index erstellen kann auch hier helfen. Eine andere Möglichkeit ist an Stelle des Filters das Range Feature zu verwenden.

Lokale Tabellen-Operationen sind sehr schnell, aber sobald sich eine zweite Anwendung mit der Datenbank verbindet wird alles sehr langsam.

Zuerst sollte geprüft werden ob es sich um ein ob es sich um ein Netzwerkproblem handelt. Da dateibasierter Zugriff extensiven Gebrauch der Netzwerkfunktionalität macht, kommt es öfters vor, dass schlechte Netzwerkperformanz erst nach Installation der TurboDB Anwendung bemerkt wird.

Das Netzwerk ist ok, editieren aber viele Anwender die Datenbank, dauert eine Operation sehr lange

Je mehr Anwender gleichzeitig auf einer Datenbank arbeiten, umso größer wird der Aufwand die Tabellensperren zu verwalten und es muss auch öfters gewartet werden bis auf die Tabellen zugegriffen werden kann. In diesem Fall ist der erste Schritt explizite Sperren für größere Operationen zu verwenden und dadurch die Anzahl der zu setzenden und wieder freizugebenden Sperren zu minimieren. Falls das nicht zum gewünschten Erfolg führt, kann noch der TurboDB Server verwendet werden.

Zu diesen eher spezifischen Tipps gibt es noch allgemeine Hinweise, die helfen können die Performanz zu steigern:

Den Flush Modus auf Fast setzen

Der Flush Mode bestimmt inwieweit die Datenbank Schreiboperationen puffert. Im Modus *Fast* ist das interne Puffer maximiert, ebenso die Geschwindigkeit. Es kann dann aber zu Datenverlust kommen, wenn die Anwendung abstürzt oder der Strom ausfällt.

Exklusiven Zugriff verwenden falls möglich

Falls die Anwendung nur für einen Anwender gedacht ist, sollte die Datenbank im Exklusivemodus betrieben werden. Damit entfällt der gesamte Aufwand der Multi-User Verwaltung.

1.3.2.7.1 Netzwerk Durchsatz und Wartezeit

Die Performanz des Netzwerk ist äußerst kritisch für die Performanz der TurboDB Anwendung, speziell dann, wenn mehrere Anwendungen auf eine Datenbank zugreifen. Probleme mit der Netzwerkkonfiguration sind der häufigste Grund für schlechten Daten-Durchsatz und sollten daher immer als erstes geprüft werden. Ein Netzwerkproblem kann vorliegen, wenn sich die Datenbank auf einem anderen Rechner befindet als die Anwendung und diese generell sehr langsam ist oder langsam wird, wenn ein zweiter Anwender auf die Datenbank zugreift.

Da Netzwerkprobleme oft sehr schwer zu identifizieren sind, bietet dataWeb das Freeware Programm *NetTest* an, das die für Datenbank-Anwendungen wichtigen Parameter des Netzwerkes misst. Sie können das Programm jederzeit beim dataWeb Support anfordern. In fünf Minuten können Sie dann herausfinden, ob Ihr Netzwerk der Schuldige ist oder nicht.

Falls das Netzwerk nicht schuld an dem Performanz-Problem ist, sind da mehrere Punkte die geprüft werden müssen:

- Prüfen Sie ob ein Anti-Viren Programm oder eine andere Software dieser Art auf TurboDB Dateien zugreift. Diese Programme tendieren dazu Tabellen-Dateien, die sich ja sehr oft ändern, nach jeder Änderung zu überprüfen und dabei den Datenbankzugriff zu verlangsamen oder ganz zu unterbinden. Anti-Viren Software sollte so konfiguriert sein, dass TurboDB Tabellen von der Überprüfung ausgeschlossen sind. Das führt zu keinen Sicherheitsloch, das es sich dabei nicht um ausführbare Dateien handelt.
- Prüfen Sie ob es für Ihre Netzwerkkarte einen aktuelleren Treiber gibt oder ob der Adapter defekt ist
- Es gibt ein bekanntes Problem mit SMB Signing, falls ein Windows XP Rechner auf einen Windows 2000 Domain Controller zugreift. Weitere Informationen und die Lösung finden Sie in der Microsoft Knowledge Base, Artikel 321098.
- Ein Hub zwischen den Datenbank Clients und dem Datenbank Fileserver blockiert manchmal den Zugriff, falls der Netzwerk-Traffik zu hoch ist. In diesem Fall sollte der Hub durch einen guten Switch ersetzt werden.

1.3.2.7.2 Sekundäre Indexe

Zusätzliche Indexe für Tabellen können Abfragen und Filter um Größenordnungen beschleunigen. Betrachten wir eine einfache Abfrage wie:

```
select * from Customers where State = 'NJ'
```

oder der vergleichbaren Filterbedingung

```
State = 'NJ'
```

Ohne Index muss TurboDB jeden Datensatz der Tabelle prüfen um diejenigen herauszufinden, die der Bedingung entsprechen. Und obwohl TurboDB wiederholte Lesevorgänge optimiert, kann die Operation für große Tabellen (einige Millionen Datensätze) einige Minuten dauern.

Wenn dagegen ein Index vorhanden ist, der mit der State-Spalte beginnt, kann die Ergebnismenge augenblicklich geliefert werden, da TurboDB die zutreffenden Datensätze direkt aussortieren kann.

Das gilt auch für Joins, eine zusätzlicher Index kann Wunder bewirken:

```
select * from Customers, Orders where Orders.CustNo = Customers.No
```

oder die Entsprechung

```
select * from Customers join Orders on Orders.CustNo = Customers.No
```

Auch hier wird ein Index über *Orders.CustNo* oder *Customers.No* die Abfrage beträchtlich beschleunigen. Ja nachdem welcher Index existiert wird TurboDB die Abfrage so abarbeiten, dass der Index verwendet werden kann. Da jedoch die *Orders*-Tabelle wahrscheinlich wesentlich größer ist als die *Customer*-Tabelle (die durchschnittliche Anzahl an Bestellungen pro Kunde ist

hoffentlich größer als eins), wird ein Index über das Feld *Orders.CustNo* hinsichtlich der Geschwindigkeit mehr bringen als ein Index über *Customer.No* (Letztere wird meistens sowieso existieren, da die Kundennummer der Primärschlüssel der Kunden-Tabelle sein wird).

Der Nachteil an Indexen ist, dass die Wartung während Änderungen (Editieren, Einfügen, Löschen) wiederum Zeit bedarf. Es muss daher die Geschwindigkeit der gesamten Anwendung im Auge behalten werden. In vielen Anwendungen sind Abfragen wesentlich häufiger als Änderungen, daher zahlen sich Indexe für die entscheidenden Anwendungsfälle meistens aus.

1.3.2.7.3 TurboSQL Statements

Einige Grundsätze für schnelle TurboSQL Abfragen:

Where und Having Klauseln sollten mit einfachen, mit and verknüpften Bedingungen beginnen

Falls es logisch machbar ist sollte die Bedingung so angeordnet sein:

```
A.a = B.a and C.b > X and (...)
```

D.h. mit einfachen Vergleichen beginnen, die für die gesamte Suchbedingung erfüllt sein müssen. Diese simplen Vergleiche sind am besten zur Optimierung geeignet. Der Optimierer versucht derartige Strukturen automatisch für die gegebene Suchbedingung zu erstellen, kann aber in manchen Fällen nicht schlau genug dazu sein.

Spalten-Bezeichner in Vergleichen separieren

Schreiben Sie

```
A.a > 2 * :ParamValue,
```

wird das besser optimiert als

```
A.a/2 > :ParamValue.
```

Entscheidend ist hier, dass die Referenz der Spalte *A.a* alleine auf der linken Seite des Vergleichs steht.

like statt Upper verwenden

Die Bedingung

```
A.a like 'USA'
```

kann optimiert werden

```
Upper(A.a) = 'USA'
```

dagegen nicht.

Left Outer Joins statt Right Outer Joins verwenden

Die Implementierung von Joins bevorzugt Left Outer Joins. Immer wenn es in einer Anwendung möglich ist, sollte

```
B left outer join A on B.a = A.a
```

verwendet werden statt

```
A right outer join B on A.a = B.a.
```

Das kann die Abfrage merklich beschleunigen. Der Optimierer macht die Konversion nicht automatisch, da sonst keine Möglichkeit bestünde, die Abfrage von Hand zu optimieren.

Reihenfolge der Tabellen in der From Klausel ändern

Diese Reihenfolge kann eine erhebliche Auswirkung auf die Geschwindigkeit der Abfrage haben. Falls Sie den Eindruck haben Ihre Abfrage könnte schneller sein, versuchen Sie die Reihenfolge der Tabellen zu variieren.

```
select * from A, B, C
where ...
```

kann viel schneller sein als

```
select * from C, B, A
where ...
```

Normalerweise wird der Optimierer die Reihenfolge der Tabellen, die nicht Teil eines Joins sind, optimal zu wählen, manchmal ist die Assistenz eines Programmierers hilfreich.

1.3.2.8 Fehlerbehandlung

Die Datenbank Engine meldet aufgetretene Fehler in Form von Exceptions.

1.3.2.8.1 Fehlerbehandlung in TurboDB Native

Es besteht immer die Möglichkeit, dass die Ausführung eines TurboDB Befehls fehlt schlägt. Die Ursachen dafür gehen von der Verwendung falscher Argumente, über Verstöße gegen die Integrität der Datenbank und Tabellensperren, bis hin zu Hardware-Fehlern und vollen Speichermedien. Tritt ein derartiges Problem auf, löst TurboDB eine Exception aus. Die genaue Klasse dieser Ausnahme hängt von der verwendeten Bibliothek ab, unabhängig davon umfasst die Fehlerinformation vier wesentliche Bestandteile:

Exception Klasse

Es gibt eine Basisklasse für alle TurboDB Exceptions und einige abgeleitete Klassen für die diversen Fehlerkategorien. Abhängig von der verwndeten Bibliothek können auch Standard-Exceptions der Bobliothek auftreten (wie *Tabelle muss geöffnet sein* in der VCL Bibliothek oder *Falscher Typ* in .NET).

Codes für die Fehlerbeschreibung

Die Basisklasse der TurboDB Exceptions verfügt über einen Fehlercode, der die fehlgeschlagene Operation bezeichnet. Im folgenden Abschnitt findet sich eine Liste der Codes für die Fehlerbeschreibung und ihre Bedeutung.

Codes für die Fehlerursache

Die Basisklasse der TurboDB Exceptions verfügt auch über eine Eigenschaft, die den Grund des Fehlers angibt. Während beispielsweise die Fehlerbeschreibung lautet Index kann nicht erstellt werden, kann der Code für die Fehlerursache entweder *Indexdatei existiert bereits* oder *Index bereits geöffnet* bedeuten. Der übernächste Abschnitt listet die Codes für die Fehlerursache und ihre Bedeutung.

Wichtige Regel zur Fehlerbehandlung

Gehen Sie nie davon aus den Grund für einen Fehler an einer bestimmten Code-Stelle zu kennen. Bei Verwendung einer Bibliothek für verbundenen Zugriff (z.B. VCL Tabellen oder das RecordSet unter .NET) darf nie folgendes gemacht werden (Es handelt sich hier um Pseudo-Code, der von allen Programmieren zu verstehen ist):

```
try
  ConnectedComponent.Post
catch
  // Post schlägt fehl, weil der Anwender ungültige Daten eingegeben hat
  ShowMessage('Die eingegebenen Daten sind ungültig, bitte
korrigieren.');
end;
```

Das dürfen Sie so nicht machen, denn es kann and dieser Stelle viele andere Gründe für eine Exception geben und einige davon sollten nicht auf die leichte Schulter genommen werden: Auf einer Ebene kann die Kapazität des Speichermediums oder der Tabelle erschöpft sein, ein Index

kann kaputt sein (wegen eines vorhergehenden Crashs), eine Datenbankdatei kann blockiert sein (beispielsweise durch einen Virenscanner). Auf einer anderen Ebene kann die Anwendung einen Fehler haben, die Komponente wurde nicht erzeugt, die Connection nicht geöffnet, der Datensatz nicht in den Editier- oder Anfügemodus versetzt, usw. In all diesen Fällen wird die gezeigte Fehlerbehandlung nicht nur Ihre Anwender behindern, sondern kann auch Auswirkungen auf die Integrität der Datenbank haben da man nicht in der Lage ist auf die eventuell fatalen Zustand zu reagieren. Eine korrekte Implementierung für obiges Beispiel sieht so aus (wieder in Pseudo-Code unter Verwendung verständlicher Bezeichner):

```
try
  ConnectedComponent.Post
catch(TurboDBError Exc)
  if Exc.ErrorCode = TdbErr_InvalidData then
    // Post schlägt fehl, weil der Anwender ungültige Daten eingegeben
hat
    ShowMessage('The data you entered is invalid, please correct it.');
  else throw;
end;
```

Jetzt wird die Meldung wirklich nur im vorhergesehenen Fall ausgegeben. Für alle anderen Ereignisse wird die Exception auf einer höheren Ebene behandelt oder zum Programmabbruch führen, was immer noch besser ist als den Fehler in einigen Fällen zu unterdrücken.

Obwohl sich diese Diskussion auf das Modell des sequentiellen Datenzugriffs konzentriert, ist die Essenz genauso auf den SQL-basierten Datenzugriff anwendbar. Obwohl Sie durch das Ausführen von SQL-Anweisungen keinen Einfluss auf die Integrität der Datenbank nehmen können, kann die Annahme den der Grund für eine Ausnahme im Voraus zu kennen, immer noch zu irreführenden Fehlermeldungen für den Benutzer und zu seltsamen Verhalten der Anwendung führen.

1.3.2.8.1.1 Codes für die Fehlerbeschreibung

Die Object Pascal Konstanten für diese Fehler Codes sind in der Unit *TdbTypes* definiert. Viele dieser Fehler haben einen zusätzlichen Reason Code.

Constant	Value	Message
TdbErr_Ok	0	Kein Fehler
TdbErr_RelationNotFound	-1	Kann Relation nicht öffnen
TdbErr_OutOfMemory	-2	Zuwenig Speicher
TdbErr_ModuleNotFound	-3	Kann Modul nicht öffnen
TdbErr_InvalidRecord	-5	Ungültiger Datensatz
TdbErr_InvalidExpression	-7	Ungültiger Ausdruck
TdbErr_InvalidField	-10	Unbekannte Tabellenspalte
TdbErr_RecNotFound	-11	Datensatz existiert nicht
TdbErr_WrongPassword	-12	Ungültiges Passwort
TdbErr_CreateFailed	-14	Relation kann nicht erzeugt werden
TdbErr_InvalidAccess	-15	Ungültiger Zugriff
TdbErr_InvalidIndex	-16	Der angegebene Index existiert nicht
TdbErr_InvalidRelation	-17	Ungültiger Relation Handle

TdbErr_InvalidIndexSpec	-18	Ungültige Index Definition
TdbErr_InvalidFunction	-19	Die Funktion kann in diesem Kontext nicht aufgerufen werden
TdbErr_DeleteFailed	-20	Kann Datensatz nicht löschen
TdbErr_RestructureFailed	-21	Restrukturierung der Relation fehlgeschlagen
TdbErr_NoAdlRltn	-22	Die Relation verfügt über keine RecordId Spalte
TdbErr_InvalidType	-23	Spalte hat nicht den korrekten Datentyp
TdbErr_WriteFailed	-25	Kann Datensatz nicht in die Tabelle schreiben
TdbErr_DuplicateRecord	-26	Datensatz verstößt gegen eindeutigen index
TdbErr_IndexInUse	-27	Der Index wird noch benutzt
TdbErr_RltnNotWritable	-30	Die Relation ist nur zum Lesen geöffnet
TdbErr_SharingConflict	-31	Der Datensatz ist von einer anderen Anwendung gesperrt
TdbErr_IndexNotAvailable	-32	Index nicht verfügbar
TdbErr_RecordIsEmpty	-33	Datensatz ist Null
TdbErr_RecordIsUnchanged	-38	Der Datensatz wurde nicht geändert
TdbErr_NoOdbcSupport	-39	Diese Edition unterstützt ODBC nicht
TdbErr_ProcNotFound	-40	Die Prozedur existiert nicht
TdbErr_InvalidModule	-41	Das Modul existiert nicht
TdbErr_DuplicateProc	-42	Die Prozedur existiert bereits
TdbErr_InvalidIdentifier	-43	Der Bezeichner ist kein gültiger TurboDB Bezeichner
TdbErr_InvalidCondition	-44	Ungültige Suchbedingung
TdbErr_RltnAlreadyOpen	-45	Die Relation ist bereits geöffnet
TdbErr_SqlFailure	-46	Fehler beim Ausführen des SQL-Befehls
TdbErr_RltnAlreadyLocked	-47	Die Relation ist bereits gesperrt
TdbErr_IndexDeleteFailed	-48	Index kann nicht gelöscht werden
TdbErr_RltnCloseFailed	-49	Relation kann nicht geschlossen werden
TdbErr_InvalidEquateJoin	-50	Fehlerhafter Equate Join
TdbErr_RltnDeleteFailed	-51	Relation kann nicht gelöscht werden
TdbErr_RelTableNoFound	-52	Relationstabelle (*.rel) nicht gefunden
TdbErr_CreateSessionFailed	-53	Session konnte nicht angelegt werden
TdbErr_CannotCreateFile	-54	Datei konnte nicht erzeugt werden
TdbErr_TransferRunning	-55	Es läuft schon ein Blob-Transfer für diesen Cursor

TdbErr_NoActiveTransfer	-56	Es läuft noch kein Blob-Transfer für diesen Cursor
TdbErr_NotInEditMode	-57	Datensatz ist nicht im Editiermodus
TdbErr_DatabaseNotFound	-58	Datenbankverzeichnis oder Datenbankdatei nicht gefunden
TdbErr_SqlSyntaxError	-59	SQL-Befehl entspricht nicht der Turbo SQL Syntax
TdbErr_CannotExecuteCommand	-60	Konnte SQL-Befehl nicht ausführen
TdbErr_EngineError	-100	Interner Fehler
TdbErr_InvalidTable	-101	Der Handle der Tabelle ist ungültig
TdbErr_InvalidStmt	-104	Der Statement Handle ist ungültig
TdbErr_ReadFailed	-201	Fehler beim Lesen aus Tabelle
TdbErr_Unspecified	-203	Unspezifizierter Fehler
TdbErr_ExportFailed	-204	Export fehlgeschlagen
TdbErr_ImportFailed	-205	Import fehlgeschlagen

1.3.2.8.1.2 Codes für die Fehlerursache

Falls der Reason Code ungleich null ist, beschreibt er die Ursache des Fehlers.

Code	Value	Description
fcCannotOpenFile	1	Datei oder Speicherobjekt kann nicht geöffnet werden
fcReadError	2	Fehler beim Lesen aus Datei oder Speicherobjekt
fcWriteError	3	Fehler beim Schreiben in Datei oder Speicherobjekt
fcIndexCreating	4	Fehler beim Erstellen eines Index
fcIndexDeleting	5	Fehler beim Löschen eines Index
fcIndexNotFound	6	Index nicht vorhanden
fcInvalidCondition	7	Fehler in Suchbedingung
fcImportError	8	Import-Fehler
fcOutputFormatError	9	Fehler in Ausgabeformat
fcCloseError	10	Fehler beim Schließen der Datei
fcIndexTooLarge	11	Größe des Index-Schlüssels darf 32KB (512 Bytes für Tabellen-Level 3 und kleiner) nicht überschreiten
fcOpeningIndexError	12	Fehler beim Öffnen des Index
fcIndexDoesNotFit	13	Index paßt nicht zur Tabelle

fcNoDataField	14	Ein Datenfeld wird im Ausdruck erwartet ist aber nicht vorhanden
fcNoFileInDir	18	Ein ungültiger Handle wurde an TurboPL übergeben
fcRecordNotFound	19	Datensatz nicht gefunden
fcModuleNotFound	20	Modul nicht gefunden
fcIndexAlreadyExists	22	Indexdatei existiert bereits
fcIndexAlreadyOpen	23	Index bereits geöffnet
fcSyntaxError	25	Syntax-Fehler
fcInvalidFileName	26	Illegaler Dateiname
fcIndexDescDamaged	28	Indexbeschreibung zerstört
fcWrongProgramFileVersion	29	Falsche Programmversion
fcDuplicateEntry	30	Doppelter Eintrag
fcCannotOpenMemoFile	31	Memo-Datei kann nicht geöffnet werden
fcMemoNotAllowedHere	32	Memo hier nicht erlaubt
fcWritingMemoFileError	33	Fehler beim Schreiben in die Memo-Datei
fcIllegalOperation	34	Anwendung hat nicht die Zugriffsrechte für diese Operation
fcTableAlreadyOpen	35	Tabelle ist bereits geöffnet
fcFileNotFound	37	Datei nicht vorhanden oder Zugriff verweigert
fcWrongKey	38	Das Passwort ist nicht korrekt
fcExpressionIncomplete	41	Ausdruck unvollständig
fcOperatorNotAllowedForOperand	42	Operator paßt nicht zu Operand
fcRealOverflow	43	Real-Überlauf
fcTypeMismatch	44	Typen stimmen nicht überein
fcIllegalCharacter	45	Zeichen an dieser Position im Ausdruck ungültig
fcInvalidNumber	46	Die Zeichenkette repräsentiert keine Zahl.
fcLogicalOperandMissing	48	Ein logischer Operand erwartet aber nicht vorhanden
fcIllegalOperand	49	Ungültiger Wert für Operand
fcUnknownIdentifier	50	Unbekannter Bezeichner
fcArrayVariableExpected	51	Array Variable erwartet
fcUnknownError	52	Unbekannter Fehler
fcTooManyVariables	53	Es wurde versucht mehr als 2048 Variablen zu

		definieren
fcEqualMissing	54	"=" im Ausdruck erwartet aber nicht vorhanden
fcNumberExpected	55	Zahl im Ausdruck erwartet aber nicht vorhanden
fcUnexpectedToken	56	Ein spezielles Merkmal im Ausdruck erwartet aber nicht vorhanden
fcInvalidColumnNumber	57	Spaltenzahl in der Quelltabelle entspricht nicht Spaltenzahl in der Zieltabelle
fcTableNameExpected	58	Tabellenname erwartet
fcTooManyColumns	59	Zu viele Tabellenspalten
fcExpressionTooComplex	60	Ausdruck zu komplex
fcTooManyIndexLevels	61	Es wurde versucht mehr als 16 Indexstufen für eine Tabelle mit Tabellen-Level < 3 zu definieren
fcNotADLTable	64	Die Tabelle hat keine AutoInc-Spalte und kann nicht als Master-Tabelle für Link- und Relationsfelder verwendet werden.
fcADLTableNotFound	65	Die über Link- oder Relationsfeld verknüpfte Tabelle kann nicht gefunden werden.
fcNoRestructureAccess	66	Das Recht die Tabellenstruktur zu ändern ist nicht vorhanden
fcTooManyIndexes	67	Die Zahl der möglichen Indexe pro Tabelle, 50 bzw. 15 für Tabellen-Level 3 und niedriger, ist überschritten
fcInvalidRange	70	Bereich nicht festgelegt
fcDuplicateIdentifier	71	Bezeichner doppelt definiert
fcTableFileAlreadyExists	72	Tabellen-Datei existiert bereits
fcTooManyLinkFields	73	Zu viele Relations-Felder
fcUpToFifteenDigitsAllowed	74	Nur 15 Stellen erlaubt
fcLineTooLong	75	Modulzeile hat mehr als 255 Buchstaben
fcMoreThanOneAutoIncField	76	Eine Tabelle kann nur eine AutoInc-Spalte haben
fcRightMarginTooSmall	79	Zu wenig Platz am rechten Rand
fcRangeOverflow	81	Bereichsüberlauf
fcIllegalCommand	82	Das TurboPL Kommando ist hier nicht erlaubt
fcInvalidAreaSequence	86	Bericht-Bereiche haben die falsche Reihenfolge
fcTableNotOpen	88	Tabelle nicht geöffnet
fcVariableExpected	89	Eine Variablenname erwartet aber nicht vorhanden
fcIndexWriting	90	Fehler beim Schreiben des Index

fcIndexReading	91	Fehler beim Lesen des Index
fcEndOfSubreportNotFound	92	Subreport begonnen aber endsub nicht vorhanden.
fcKeywordNotAllowed	93	Das Schlüsselwort ist im aktuellen Bereich nicht erlaubt
fcTooManyTables	94	Es können maximal 254 Tabellen pro Session geöffnet werden (62 für Tabellen-Level 3 und niedriger)
fcIndexDamaged	95	Der Index ist beschädigt
fcUntilExpected	96	Repeat Schleife begonnen aber until nicht vorhanden
fcEndExpected	97	if oder while Block begonnen aber end nicht vorhanden
fcInvalidIndexSpec	98	Ungültige Index Definition
fcOutOfMemory	99	Speicher reicht nicht aus
fcDemoVersion	100	Demoversion erlaubt nicht mehr Datensätze
fcLocalProceduresNotPermitted	101	TurboPL unterstützt keine verschachtelten
fcEndProcExpected	102	Eine Prozedur wurde begonnen aber endproc nicht vorhanden
fcTableInUse	103	Die Operation kann nicht abgeschlossen werden, weil die Tabelle von einer anderen Anwendung/Session verwendet wird
fcTableIsLocked	104	Die Tabelle ist von einer anderen Anwendung/Session gesperrt
fcRecordEdited	105	Der Datensatz ist von einer anderen Anwendung/Session gesperrt
fcErrorInLogin	106	Die Tabelle kann nicht geöffnet werden, weil die Verwaltung der Netzdateien nicht erfolgreich war
fcInvalidConnectionId	107	Ungültige Connection Id
fcUnknownLockError	109	Unbekannter Netzwerkfehler
fcLockingError	110	Dateisperren werden vom Betriebssystem nicht ausreichend unterstützt
fcCannotOpenBlob	111	Blob Datei kann nicht geöffnet werden
fcMemoDamaged	112	Memo Datei ist beschädigt
fcBlobDamaged	113	Blob Datei ist beschädigt
fcUserAbort	114	Die Operation wurde vom Benutzer abgebrochen
fcNoWriteAccess	115	Kein Schreibrecht auf Datei
fcIndexInUse	116	Die Operation kann nicht abgeschlossen werden, weil der Index von einer anderen

		Anwendung/Session benutzt wird
fcUnsupportedTableFeature	117	Das Schema der Tabellen ist ungültig, weil es ein Merkmal verwendet, das nur von einem höheren Tabellen-Level unterstützt wird
fcErrorInExecutionExternalProcedure	118	Fatal error while executing external procedure
fcExternalProcedureNotFound	119	Fataler Fehler beim Aufruf der externen Prozedur
fcNoOdbcSupport	120	ODBC konnte nicht initialisiert werden
fcCannotOpenODBCDataSource	121	Die ODBC-Datenquelle kann nicht geöffnet werden
fcErrorInQuery	122	Bei der SQL-Abfrage ist ein Fehler aufgetreten
fcException	123	Ausnahme in TurboDB Engine
fcOldDatVersion	124	Veraltete Version der Tabellen-Datei
fcCannotAssignToConst	125	Einer Konstanten kann nichts zugewiesen werden
fcObjectExpected	126	Links von . muss ein Objekt stehen
fcArrayTooLarge	127	Array hat zuviele Elemente (bis zu 2 GB möglich)
fcInvalidFullTextSearch	128	Die Volltext-Suchbedingung enthält einen Fehler
fcUnknownDBaseFormat	129	Die Version der dBase-Datei ist unbekannt
fcSharingViolation	130	Die Datei wird noch von einem anderen Anwender benutzt
fcUnknownClass	132	Unbekannte Klasse
fcInvalidObject	133	Ungültige Objekt-Referenz
fcTableCorrupt	134	Dateikopf der Tabelle ist beschädigt
fcErrorInUICall	135	Fehler beim Aufruf einer Bibliotheks-Routine
fcInvalidMember	136	Unbekanntes Klassen-Element
fcInvalidDate	137	Die Zeichenkette repräsentiert kein gültiges Datum
fcLangDriverNotFound	138	Sprachtreiber wurde nicht gefunden oder entspricht nicht der Spezifikation
fcInvalidAggregate	139	Das Argument einer Aggregations-Funktion muss nummerisch sein
fcInvalidTime	140	keine gültige Zeitangabe
fcNoCreatePermission	141	Kein Recht, Datei zu erstellen
fcNoReadPermission	142	Kein Recht, Datei zu lesen
fcFieldSizeIsZero	143	Feld hat die Größe Null

fcUnknownFieldType	144	Unbekannter Feldtyp
fcIndicationMissing	145	AutoInc-Feld hat keine Anzeigeinformation
fcInvalidJoin	146	Ungültiger Join
fcDifferentLangDriver	147	Alle Tabellen einer Sitzung müssen denselben Sprachtreiber benutzen
fcInvalidStrValue	148	kein gültiger Wert fr das Feld
fcNoIndexPermission	149	Kein Recht, einen Index zu erstellen
fcValueListTooLong	150	Zu viele Werte in der Liste
fcSingleTableJoin	151	Equate join hat genau eine Tabelle auf jeder Seite des =
fcNotDBaseCompatible	152	DBase III-Dateien können diese Datenstruktur nicht aufnehmen
fcExternalTableError	153	Fehler in externer Tabelle
fcRelTableNotFound	154	Mindestens eine Relationstabelle konnte nicht geöffnet werden
fcDeletionNotComplete	155	Eine oder mehrere Dateien konnten nicht gelöscht werden
fcBlobFileTooLarge	156	Die Memo/Blob Datei überschreitet die maximale Größe
fcNoAutoIncModify	157	AutoInc Feld kann nicht geändert werden
fcInvalidDateTime	158	Ungültiger Zeitstempel
fcLangDriverNotSupported	159	Sprachtreiber wird vom Betriessystem nicht unterstützt
fcNoWritePermission	160	Schreibrecht für Datei nicht vorhanden
fcExpressionHasNoValue	161	Der Ausdruck hat keinen Wert, auch nicht Null
fcWrongFieldType	162	Der Spaltentyp ist nicht wie erwartet
fcNextExpected	163	for-loop wurde begonnen aber next nicht vorhanden
fcUnknownType	164	Der Typname ist unbekannt
fcCOMError	165	Fehler beim Aufruf eines COM Interface
fcCOMError	166	Module verwendet sich direkt oder indirekt selbst
fcConstraintViolated	167	Datensatz verletzt eine Gültigkeitsbedingung
fcBlobFieldExpected	168	Spaltenreferenz auf Blob erwartet aber nicht vorhanden
fcMemoFieldExpected	169	Spaltenreferenz auf Memo erwartet aber nicht vorhanden
fcLocksPresent	170	Transaktion kann nicht gestartet werden solange

		Sperren auf der Tabelle sind
fcCapacityLimit	171	Einer der Tabellenindexe hat seine maximale Kapazität erreicht und muss neu erstellt werden
fcIncompleteGroupBy	172	Group By Klausel nicht vollständig
fcInvalidColumnName	173	Zeichenkette ist kein gültiger Spaltenbezeichner
fcIdentifierAmbigous	174	Bezeichner nicht eindeutig
fcTableStillBusy	175	Tabelle wird noch von dieser Session verwendet
fcInvalidIndexName	176	Zeichenkette ist kein gültiger Indexname
fcNotEnoughArguments	177	Prozedur hat nicht genügend Parameter
fcDivisionByZero	178	Division durch Null aufgetreten
fcNoParentRowFound	179	Fremdschlüsselbedingung kann nicht erfüllt werden
fcIntegrityViolated	180	Operation würde Fremdschlüsselbedingung verletzen
fcRecordNotEditing	181	Datensatz ist gesperrt
fcOutOfTable	182	Position des Cursors ist außerhalb der Tabelle
fcTransactionRunning	183	Es läuft bereits eine Transaktion
fcParentTableNotFound	184	Parent Tabelle für eine Fremdschlüsselbedingung ncht gefunden
fcIncompatibleLockFile	185	Anwendungen inkompatibler TurboDB Versionen verwenden die Tabelle

1.3.2.9 Verschiedenes

Datenbank-Dateien beschreibt welche physikalischen Datenbankdateien TurboDB kennt und wozu sie gut sind.

Datensicherheit behandelt die verschiedenen Methoden um Ihre Daten zu schützen.

Sprachunabhängigkeit skizziert den Weg wie Sie Ihre TurboDB Anwendung an spezielle Gebietsschemata anpassen.

1.3.2.9.1 Datenbank-Dateien

Hier werden die Dateien beschrieben, die zu einer TurboDB Datenbank gehören. Sie werden nach Ihrer Erweiterung unterschieden. Wenn Sie mit einer SingleFile-Datenbank arbeiten, können Sie diese Erweiterungen nicht direkt sehen. Verwenden Sie dataweb Compound File Explorer, dann sehen Sie, dass die Datenbank-Datei Speicherobjekte (Dateien) mit denselben Erweiterungen enthält.

Level 6	Level 1-5	Beschreibung
tdbd	*tdbd*	*tdbd* steht für TurboDB Datenbank. Diese Datei enthält alle Tabellen und Indexe, die zu einer Datenbank gehören, wenn die Datenbank als

		Single-File-Datenbank angelegt wurde. In diesem Fall sind im Dateisystem des Betriebssystems keine *dat, mmo, blb, rel, id, in?* und *ind*-Dateien zu sehen, weil die entsprechenden Daten alle innerhalb der tdbd-Datei abgelegt sind.
tdbt	*dat, rel*	Beinhalten die Datenbanktabellen, also die Datensätze. *Rel* Dateien sind spezielle Datenbanktabellen, die automatisch erzeugt werden um n:m Relationen zu implementieren. Diese Dateien sind mit einer Anwendung weiterzugeben.
tdbm, *tdbb*	*mmo*/*blb*	Das sind die Memo und Blob Dateien, die für jede Tabelle existieren, die über mindestens ein Memo bzw. mindestens ein Blob Feld verfügt. Eine solche Datei beinhaltet alle die Daten aller Memo oder Blob Felder der Tabelle. Memo oder Blob Dateien müssen mit einer Anwendung weitergegeben werden.
tdbi	*ind*	Anwederspezifische Indexe. Jede ind Datei enthält einen Index. Diese Dateien sind mit einer Anwendung weiterzugeben.
tdbi	*id*, *inr*, *in?*	Automatisch erstellte Indexe für den Primärindex, falls definiert. Die id Datei ist ein Index über die Standard Sortierordnung and die inr Datei ist der Index über das AutoInc Feld. Diese Dateien sind mit einer Anwendung weiterzugeben.
tdbf	*fti*	Volltext-Index
tdbl, *tdbv*	*net*, *rnt*, *mov*, *rmv*	Das sind Netzwerkverwaltungsdateien und existieren für jede Tabelle, die im Shared Modus geöffnet wurde. Geben Sie diese Dateien nicht mit Ihrer Anwendung zum Kunden. Sie enthalten nur dynamisch erzeugte Information. Falls die Anwendung abstürzt oder während dem Debuggen abgebrochen wird, bleiben diese Dateien auf der Festplatte zurück und können beim erneuten Start der Anwendung zu Fehlermeldung, wie "Tabelle wird von einer anderen Anwendung benutzt", führen. Öffnen Sie in diesem Fall die Datenbank und die betreffenden Tabellen mit TurboDB Viewer oder einer beliebigen anderen TurboDB Anwendung. Beim Schließen der Anwendung werden die Netzwerkverwaltungsdateien gelöscht.
tdbr	*tra*, *rtr*	Diese Dateien sind die Redo-Logs für Transaktionen. Während einer Transaktion, gibt es eine tra Datei (rtr für Relations Tabellen) für jede Tabelle, die während einer Transaktion geändert wird. Nach Ende der Transaction werden diese Dateien gelöscht. Falls eine dieser Dateien ohne laufende Transaktion in der Datenbank zu sehen ist bedeutet dies, dass eine Anwendung während einer Transaktion gestorben ist. Löschen Sie die Redo-Logs nicht. Die Anwendung, die als nächstes die Tabellen verwendet wird ein Rollback für die unterbrochene Transaktion ausführen, um die Integrität der Datenbank zu gewährleisten.
		Temporäre Tabllen haben Zufallsnamen wie jzbgopqw.dat und die temporären Indexe heißen entsprechend. Diese Dateien werden für gewöhnlich im temporären Windows Verzeichnis oder im Home Verzeichnis unter Linux abgespeichert. Sie können aber auch andere Verzeichnisse bestimmen indem Sie die Eigenschaft PrivateDir der TTdbDatabase Komponente setzen.

1.3.2.9.2 Datensicherheit

Normalerweise kann Ihre TurboDB Datenbanktabelle von jeder Person gelesen werden, die Zugriff auf die Datei hat und die über ein Werkzeug verfügt, mit dem TurboDB Dateien geöffnet werden können. Um das zu verhindern, können Sie Ihre Tabellen mit einem Passwort schützen. Alle TurboDB Tools verlangen Sie dieses Passwort und werden den Inhalt der Tabelle nicht anzeigen, bis der Anwender das richtige Passwort eingegeben hat.

Obwohl das in vielen Fällen bereits ausreichend sein kann, ist es kein wirklicher Schutz Ihrer Daten. Wie auch andere dateibasierte Datenbanken (z.B, Access, dBase, Paradox) speichert TurboDB die Daten direkt in den Datenbankdateien. Das bedeutet, dass der Inhalt mit einem Binäreditor oder sogar einem beliebigen Texteditor eingesehen werden kann. Dies trifft auch zu, wenn Sie Ihre Tabelle mit einem Passwort versehen, da der Passwortschutz nicht die Art

verändert, mit der die Daten in der Datei abgelegt werden. Falls Sie Ihre Daten effektiv vor unautorisierten Blicken schützen wollen, bietet TurboDB verschiedene Verschlüsselungs-Algprithmen an, die jeden Datensatz verschlüsseln bevor er in die Tabellen-Datei geschrieben wird.

Die klassische TurboDB-Verschlüsselung basiert auf einem 32 Bit Schlüssel. Wie Sie sicher wissen ist ein 32 Bit Schlüssel nicht sicher genug für Banking oder andere Hochsicherheits-Angelegenheiten. Für die meisten Anforderungen wird dieser Sicherheitslevel aber völlig ausreichend sein und ein kurzer Schlüssel sorgt für schnelle Datenbankaktionen.

Wenn Sie sicherere Verschlüsselung für Ihre Daten benötigen, können Sie einen der starken Verschlüsselungalgorithmen verwenden, die in TurboDB angeboten werden. Diese Algorithmen schützen Ihre Daten von jedem, der nicht den Schlüssel kennt. Zum derzeitigem Stand der Verschlüsselungtechnologie, können diese Chiffre sogar mit hoch entwickelten Dekodierungalgorithmen und Computerhardware nicht entschlüsselt werden.

Die Verschlüsselungmethode kann auf Datenbankebene (für verwaltete Datenbanken) oder auf Tabellenebene definiert werden. Wenn Sie die Verschlüsselung auf Datenbankebene definieren, müssen Sie die Verschlüsselungmethode und das Kennwort nur einmal festlegen, wenn Sie die Datenbank erstellen. Der Benutzer muss das Kennwort nur einmal für alle Tabellen der Datenbank eingeben. Folglich ist dies die empfohlene Vorgehensweise.

Frühere Versionen von TurboDB verlangten beides, ein Passwort und eine Code genannte 32-bit Zahl, um eine verschlüsselte Tabelle zu öffnen. Aktuelle Versionen erfordern nur eine Zeichenkette, das Passwort. Um kompatibel zu sein wird die frühere Kombination aus Passwort und Code zu einer Zeichenkette in diesem Format verknüpft: <Passwort>;<Code>. So werden das Passwort secret und der Code -3871 jetzt als das Password secret;-3871 eingegeben.

Hier eine Liste der verfügbaren Sicherheitseinstellungen. Diese werden in den Komponenten-Bibliotheken durch den Aufzählungswert für den Verschlüsselungstyp (encryption method) angegeben.

Name	Beschreibung	Schlüssel
Default	Für Tabellen in verwalteten Datenbanken: Bezieht die Einstellungen zur Verschlüsselung von der Datenbank. Andere Tabellen und Datenbanken: Keine Verschlüsselung	Siehe jeweilige Zeile
None	Weder Verschlüsselung noch Passwortschutz	-
Protection	Die Tabelle ist nicht verschlüsselt aber mit Passwort geschützt.	Das Passwort, z.B. 3Huv
Classic	Die Tabelle ist mit einem 32 Bit Schlüssel verschlüsselt und mit Passwort geschützt.	Das Passwort und der nummerische Schlüssel durch Strichpunkt getrennt, z.B. 3Huv;97809878
Fast	Die Tabelle ist mit einer sehr schnellen 32-Bit Chiffre verschlüsselt. Ausreichend für vielen Anwendungen aber nicht 100% sicher.	Ein alphanummerisches Passwort bis zu 40 Zeichen, z.B. 3Huv
Blowfish	Verschlüsselung mit dem bekannten Blowfish Algorithmus, wobei ein 128 Bit Schlüssel verwendet wird.	Ein alphanummerisches Passwort bis zu 40 Zeichen, z.B. 3Huv
Rijndael	Verschlüsselung mit dem bekannten Rijndael Algorithmus, wobei ein 128 Bit Schlüssel verwendet wird. Bekannt unter Advanced Encryption Standard (AES).	Ein alphanummerisches Passwort bis zu 40 Zeichen, z.B. 3Huv

AES	Wie Rijndael.	Wie Rijndael

1.3.3 TurboPL Guide

TurboDB Engine verfügt über einen Satz nativer Funktionen. Diese Funktionen wurden bis Tabellen-Level 4 zur Formulierung von Gültigkeitsbedingungen, berechneten Felder und Indexen und Vorgabewerten verwendet. Ab Tabellen-Level 5 wird diese Funktionalität auch mit TurboSQL abgedeckt, TurboPL kann aber trotzdem noch eingesetzt werden. Soll eine TurboPL Funktion in der Formel eines berechneten Feldes oder eines berechneten Indexes verwenden werden, muss ein @-Zeichen vorangestellt werden.

Beispiel: *@RightStr(Spalte1)* verwendet den TurboPL Interpreter anstelle des TurboSQL Interpreters zur Auswertung der Formel. Dies ist wichtig für die Rückwärtskompatibilität und wird für Tabellen ab Level 5 nicht empfohlen.

- Operatoren und Funktionen
- Suchbedingungen

Siehe auch

TurboSQL Guide

1.3.3.1 Operatoren und Funktionen

- Arithmetische Operatoren und Funktionen
- String Operatoren und Funktionen
- Datum und Zeit Operatoren und Funktionen
- Sonstige Operatoren und Funktionen

1.3.3.1.1 TurboPL Arithmetische Operatoren und Funktionen

Die folgenden arithmetischen Operatoren und Funktionen können in TurboPL Ausdrücken eingesetzt werden. Sie werden nicht mehr für TurboSQL empfohlen

Operatoren

+	Addition
-	Subtraktion
*	Multiplikation
/	Fließkomma Division
div	Ganzzahl Division
mod	Modulo

Vergleiche

<	kleiner
<=	kleiner oder gleich
=	gleich
>=	größer oder gleich
>	größer
less	kleiner
is	gleich

greater	größer
<>	ungleich
from...upto	testet auf einen Bereich
in [...]	testet, ob ein Element in einer Menge enthalten ist

Functions

Abs(X: Real): Real	Liefert den absoluten Wert des Arguments, X.
ArcTan(X: Real): Real	Liefert den ArcTan der gegebenen Zahl X.
Cos(X: Real): Real	Liefert den Cosinus des Winkels X, im Bogenmaß.
Exp(X: Real): Real	Liefert die Exponentialfunktion der übergebenen Zahl.
Frac(X: Real): Real	Ermittelt den gebrochenzahligen Anteil der Zahl.
Int(X: Real): Integer	Liefert den ganzzahligen Anteil einer Zahl.
Log(X: Real): Real	Berechnet den natürlichen Logarithmus der Zahl.
Round(X: Real; [Precision: Integer]): Real	Rundet die angegebenen Zahl auf eine bestimmt Zahl an Nachkommastellen.
Sin(X: Real): Real	Berechnet den Sinus einer Zahl.
Sqrt(X: Real): Real	Berechnet die Quadratwurzel einer Zahl.

Kompatibilität

TurboPL wird nur zur Rückwärtskompatibilität in Tabellen bis Level 4 unterstützt.

1.3.3.1.2 TurboPL String Operatoren und Funktionen

Diese String Operatoren und Funktionen können in TurboPL Ausdrücken eingesetzt werden. Sie werden nicht mehr für TurboSQL empfohlen

Operatoren

+	String-Addition, z.B. EMPLOYEES.FirstName + ' ' + EMPLOYEES.LastName
[]	Zugriff auf einzelne Zeiche, z.B. EMPLOYEES.FirstName[1] + '. ' + EMPLOYEES.LastName

Vergleiche

<	kleiner
<=	kleiner oder gleich
=	gleich
>=	größer oder gleich
>	größer
less	kleiner
equal	gleich
greater	größer

<>	ungleich
has	sucht einen String innerhalb eines anderen mit Berücksichtigung von Groß- und Kleinschreibung, z.B: 'John Smith' has 'Sm'
like	vergleicht ohne Berücksichtigung von Groß/Klein-Schreibung mit einer Maske, die auch Joker-Zeichen enthalten kann, z.B: 'Smith' like 'sMIth', 'Smith' like 'Sm*', 'Smith' like 'Smit?' (alle wahr)
in [..]	testet, ob ein Element in einer Menge enthalten ist

Funktionen

Argumente in eckigen Klammern sind optional.

Asc(C: String): Integer	ANSI-Code des Zeichens.
Chr(N: Integer): String	Zeichen mit übergebenen ANSI-Code.
Exchange(Source, From, To: String): String	In der *Source* werden alle Vorkommen von From durch *To* ersetzt
FillStr(Source, Filler: String; Len: Integer): String	Füllt die Zeichenkette *Source* mit *Filler* bis zur durch *Length* gegebenen Länge und gibt das Ergebnis zurück.
LeftStr(Source: String; Len: Integer): String	Gibt die linken *Len* Zeichen von *Source* zurück.
Length(Source: String)	Liefert die Anzahl der Zeichen in *Source*.
Lower(Source: String): String	Liefert die Zeichenkette in Kleinbuchstaben konvertiert.
LTrim(Source: String): String	Gibt *Source* ohne führende Leerzeichen zurück.
MemoStr(Memo: MemoField [; Len: Integer]): String	Liefert die ersten 255 Zeichen des Inhalts eines *Memofeldes* des aktuellen Datensatzes.
NTimes(Source: String; Count: Integer): String	Liefert einen String, in dem die mit *Source* gegebene Zeichenkette *Count* mal wiederholt wird.
RealVal(Str: String): Real	Berechnet den nummerischen Wert eines String-Ausdrucks.
Pos(SubStr, Source: String): Integer	Liefert die Position der Zeichenkette *SubStr* in Source oder 0, falls *SubStr* nicht in *Source* enthalten ist.
RightStr(Source: String; Len: Integer)	Liefert die *Len* letzten Zeichen von Source.
RTrim(Source: String): String	Liefert *Source* ohne abschließende Leerzeichen.
Scan(SubStr, Source: String): Real	Zählt, wie oft *SubStr* in *Source* vorkommt.
Str(Num: Real[; Width, Precision: Real]): String	Wandelt *Num* in eine Zeichenkette um. *Width* gibt die Länge der Zeichenkette an, innerhalb derer die Zahl rechtsbündig dargestellt wird. Reicht die Länge zur Darstellung der Zahl nicht aus, wird die Zeichenkette automatisch soweit nötig erweitert. *Precision* bestimmt die Anzahl der Nachkommastellen. Die alphanummerischen Werte eines Aufzählungsfeldes (dtEnum) (siehe Datentypen für Tabellenspalten) können abenfalls mit *Str* ermittelt werden.
Upper(Source: String): String	Liefert den übergebenen String in Großbuchstaben.
NewGuid: String	Liefert eine Zeichenkette, die einen neuen Globally Unique Identifier bezeichnet.

Kompatibilität

TurboPL wird nur zur Rückwärtskompatibilität in Tabellen bis Level 4 unterstützt.

1.3.3.1.3 TurboPL Datum- und Zeit-Operatoren und Funktionen

Diese Datum und Zeit Operatoren und Funktionen können in TurboPL Ausdrücken eingesetzt werden. Sie werden nicht mehr für TurboSQL empfohlen

Vergleiche

Alle nummerischen Operatoren können genauso auch für Datum- und Zeit-Werte verwendet werden. Z.B. Datum 1 > Datum2.

Berechnungen

Sie können Zeitspannen einem Datum hinzufügen oder von einem Datum subtrahieren und Datums-Werte voneinander subtrahieren, um die Zeitspanne zu erhalten. Eine Zeitspanne ist eine Fließkommazahl, die eine Anzahl an Tagen repräsentiert (einschließlich des Nachkommaanteils für die Tageszeit) beim Rechnen mit Datums- und DateTime-Werten oder die Anzahl der Minuten (einschließlich des Nachkommaanteils für Sekunden und Millisekunden), wenn mit Zeit-Werten gerechnet wird.

Falls Time1 und Time2 Zeit-Variablen, Date1 und Date2 Datum-Variablen, DateTime1 und DateTime2 Variablen vom Typ DateTime und TimeSpan1 und TimeSpan2 Real Variablen sind, sind die folgenden Ausdrücke sinnvoll:

```
Time2 - Time1
Time2 - TimeSpan1
Time1 + TimeSpan2
Date2 - Date1
Date2 - TimeSpan1
Date2 + TimeSpan2
DateTime2 - DateTime1
DateTime2 - TimeSpan1
DateTime2 - TimeSpan2
```

Über die nummerischen Operatoren und Funktionen hinaus gibt es auch noch spezielle Datum- und Zeit-Funktionen:

CombineDateTime

CombineDateTime(ADate: Date; ATime: Time): DateTime

Kombiniert ein Datum und eine Zeit zu einem DateTime-Wert

DateStr

DateStr(ADateTime: DateTime): String

Konvertiert ein Datum oder einen Zeitstempel in eine Zeichenkette, die dem aktuell eingestellten Datumsformat entspricht.

DateTimeStr

DateTimeStr(ADateTime: DateTime; TimePrecision: Integer): String

Konvertiert einen Zeitstempel in eine Zeichenkette, die dem aktuell eingestellten Datumsformat entspricht. *TimePrecision* bestimmt dabei die Genauigkeit (minute = 2, second = 3, millisecond = 4)

Day

Day(ADate: DateTime): Integer

Extrahiert den Tag aus einem Datum.

DayOfWeek

DayOfWeek(ADateTime: DateTime): String

Liefert den Namen des Wochentags für das eingestellte Gebietsschema.(bspw. Mittwoch)

Hour

Hour(ADate: DateTime): Integer

Extrahiert die Stunden aus einem Time- oder DateTime-Wert.

Millisecond

Millisecond(ADate: DateTime): Integer

Extrahiert den Millisekunden-Anteil aus einem Time- oder DateTime-Wert.

Minute

Minute(ADate: DateTime): Integer

Extrahiert die Minuten aus einem Time- oder DateTime-Wert.

Month

Month(ADate: DateTime): Integer

Extrahiert den Monat aus dem Datum.

Now

Now: Time

Liefert die aktuelle Uhrzeit.

Second

Second(ADate: DateTime): Integer

Extrahiert die Sekunden aus einem Time- oder DateTime-Wert.

TimeStr

TimeStr(ATime: Time): String

In Abhängigkeit vom einstellten Gebietsschema wird eine Zeit oder einen Zeitstempel in einen String konvertiert.

Today

Today: Date

Liefert das aktuelle Datum.

Week

Week(ADate: DateTime): Integer

Ermittelt die Wochen-Nummer für das Datum.

WeekDayNo

WeekDayNo(ADateTime: DateTime): Integer

Liefert den Wochentag als Nummer zwischen 1 (Montag) und 7 (Sonntag)

Year

Year(ADate: DateTime): Integer

Extrahiert das Jahr aus einem Datum.

Kompatibilität

TurboPL wird nur zur Rückwärtskompatibilität in Tabellen bis Level 4 unterstützt.

1.3.3.1.4 TurboPL Sonstige Operatoren und Funktionen

Darüberhinaus gibt es weitere Operatoren und Funktionen, die in TurboPL Ausdrücken eingesetzt werden können. Sie werden nicht mehr für TurboSQL empfohlen

Funktion	Beschreibung
HexStr(Value: Integer [; Digits: Integer])	Liefert die hexadezimale Darstellung einer Zahl.
CurrentRecordId(TableName)	Gibt den letzten vergebenen Wert für die AutoInc-Nummer der Tabelle zurück. Mit dieser Funktion kann man in einem einzigen zusammengesetzen Statement verknüpfte Datensätze in mehrere Tabellen eintragen.

Kompatibilität

TurboPL wird nur zur Rückwärtskompatibilität in Tabellen bis Level 4 unterstützt.

1.3.3.2 Suchbedingungen

- Filter Suchbedingungen
- Volltext Suchbedingungen

1.3.3.2.1 Filter Suchbedingungen

Filter-Suchbedingungen sind boolsche Ausdrücke, wie sie auch in SQL oder Pascal zum Einsatz kommen. Sie werden beispielsweise verwendet um mit dem TurboDB Viewer oder der VCL-Komponente *TTdbTable* Gültigkeitsbedingungen für Tabellen anzugeben. Die Syntax und die verfügbaren Optionen sind identisch mit den in TurboSQL möglichen Bedingungen in Where-Klauseln mit zwei Ausnahmen, die zur besseren Kompatibilität mit BDE-Filtern eingeführt wurden:

- Datum-, Zeit- und numerische Formate basieren auf den lokalen Einstellungen des Systems
- * und ? sind als Joker ebenso erlaubt wie % und _.
- Mengen werden in eckigen statt in runden Klammern geschrieben

Beispiele (Name, Betrag, Betrag1, Betrag2 und Geburtstag sind Tabellenspalten)

```
Name = 'Schmidt'
Name like 'Schmi*'
Name like 'Schmi%'
Name like 'Schmid?'
Name like 'Schmid_'
Name has 'mid'
LeftStr(Name, 2) = 'Sc'
Length(Name) > 4
Betrag = 13546,45
Betrag < 13546,46
Betrag < 345,67 or Betrag > 567,89 (Das Dezimaltrennzeichen kann von den lokalen Einstellungen des Systems abhängen.)
```

```
Betrag1 * 0,3 > Betrag2 * 0,8
```

```
Betrag is not null (alle Datensätze, die für Betrag keinen Wert haben)
```

```
Geburtstag = '20.4.1962' (Das Datums-Format kann von den lokalen
Einstellungen des Systems abhängen.)
```

```
Geburtstag < '20.4.1962'
```

```
Geburtstag between '1.4.1962' and '30.4.1962'
```

```
Year(Geburtstag) = 1962
```

```
Date-of-birth is null  (alle Datensätze, die für Geburtstag keinen Wert
haben)
```

Regeln für Anführungszeichen und Escaping

Zeichenketten werden in einfache Anführungszeichen eingeschlossen. Anführungszeichen innerhalb von Zeichenketten, die von denselben Anführungszeichen umschlossen sind, sind zu duplizieren:

```
'My "quote"' -> My "quote"
```

```
'My ''quote''' -> My 'quote'
```

Falls der Name einer Tabellen-Spalte, auch ein TurboSQL Schlüsselwort ist, muss der Bezeichner entweder in doppelten Anführungszeichen oder in eckigen Klammern geschrieben werden:

```
Length("Password") > 8
```

```
Week([Date]) = 18
```

TurboDB bietet leistungsstarke Funktionen und Operatoren zum Einsatz in Suchbedingungen, z.B. *like*, *from...upto*, *LeftStr*, *Year* und viele mehr. In Operatoren und Funktionen finden Sie eine komplette Referenz. Bedingungen können mit den logischen Operatoren *and*, *or* und *not* kombiniert werden.

Kompatibilität

TurboPL wird nur zur Rückwärtskompatibilität in Tabellen bis Level 4 unterstützt.

1.3.3.2.2 Volltext Suchbedingungen

Volltext Suchbedingungen werden im TurboSQL CONTAINS Prädikat und in der VCL *TTdbTable.WordFilter* Property verwendet. Eine Volltext-Suchbedingung ist im Wesentlichen eine Liste von Schlüsselwörtern, voneinander getrennt durch "+", "," oder "-". Diese Zeichen stehen für:

, oder Leerzeichen	beide Schlüsselwörter müssen im Datensatz vorhanden sein (und Verknüpfung)
+ oder /	eines der Schlüsselwörter muss im Datensatz zu finden sein (oder Verknüpfung)
-	Das Schlüsselwort darf nicht im Datensatz vorhanden sein (nicht Operator)

Die alternativen Zeichen (Leerzeichen und /) sind nur ab Tabellen-Level 4 verfügbar. Schlüsselwörter dürfen die Joker "?" und "*" enthalten um entweder ein beliebiges Zeichen oder eine beliebige Zeichenfolge zu repräsentieren.

Beispiele

Database	Findet *Database*, *database*, *dataBase*, ...
Database*	Findet *database*, *Databases*, *DatabaseDriver*, ...
Data?ase	Findet *Database*, *dataCase*, ...
Database, Driver	Im Datensatz müssen die Wörter *Database* and *Driver* vorkommen
Database Driver	Wie oben, für Tabellen-Level 4
Database, Driver, ODBC	Im Datensatz müssen die Wörter *Database*, *Driver* and *ODBC* vorkommen

Database Driver ODBC	Wie oben, für Tabellen-Level 4
Database + Driver	Im Datensatz müssen das Wort *Database* oder das Wort *Driver* oder beides vorkommen
Database/Driver	Wie oben, für Tabellen-Level 4
Database + Driver + ODBC	Im Datensatz muss entweder *Database* oder *Driver* oder *ODBC* vorkommen
Database/Driver/ODBC	Wie oben, für Tabellen-Level 4
Database Driver ODBC/OLE	Im Datensatz müssen die Wörter *Database* und *Driver* und entweder *ODBC* oder *OLE* vorkommen
-Database	Im Datensatz darf das Wort *Database* nicht vorkommen
Database - Driver	Im Datensatz muss das Wort *Database* vorkommen, *Driver darf* dagegen nicht enthalten sein

Kompatibilität

TurboPL wird nur zur Rückwärtskompatibilität in Tabellen bis Level 4 unterstützt.

1.3.4 TurboSQL Guide

Turbo SQL ist eine Untermenge von SQL 92, die in der MS ODBC Spezifikation enthaltene SQL Definition und ist sehr ähnlich dem in der Embarcadero Database Engine enthaltenem Local SQL.

Konventionen

- Tabellennamen
- Spaltennamen
- String Literale
- Datumsformate
- Zeitformate
- DateTime Format
- Boolsche Konstanten
- Tabellenkorrelationsnamen
- Spaltenkorrelationsnamen
- Parameter
- Eingebettete Kommentare

Data Manipulation Language

- Überblick
- DELETE Statement
- FROM Klausel
- GROUP BY Klausel
- INSERT Statement
- ORDER BY Klausel
- SELECT Statement
- UPDATE Statement
- WHERE Klausel

Data Definition Language

- Überblick
- CREATE TABLE Kommando
- ALTER TABLE Kommando
- CREATE INDEX Kommando
- DROP Kommando
- Datentypen für Tabellenspalten

Programmiersprache

- Überblick
- CALL Statement
- CREATE FUNCTION Statement
- CREATE PROCEDURE Statement
- CREATE AGGREGATE Statement
- DROP FUNCTION/PROCEDURE/AGGREGATE Statement
- DECLARE Statement
- IF Statement
- SET Statement
- WHILE Statement
- Parameteraustausch mit .NET Assemblies

1.3.4.1 TurboSQL vs. Local SQL

TurboSQL unterscheidet sich in einigen Punkten von Embarcadero's Local SQL:

- In TurboSQL können Sie Datums, Zeit und Datetime Literale ohne Anführungszeichen angeben, das Format ist tt.mm.jjjj und ss:mm und tt.mm.jjjj_hh:mm:ss.ms
- Mit TurboDB SQL können mehrere durch Semikolon getrennte Kommandos in einem Statement zusammengefasst werden.
- TurboDB kann Spalten bestehender Tabellen mit Hilfe des ALTER TABLE Kommandos umbenennen und ändern werden.

1.3.4.2 Konventionen

1.3.4.2.1 Tabellennamen

TurboDB beschränkt wie der ANSI-Standard-SQL alle Tabellennamen auf ein einzelnes Wort, das sich aus alphanumerischen Zeichen und dem Unterstrich, "_", zusammensetzt.

```
SELECT *
FROM customer
```

TurboSQL unterstützt vollständige Datei- und Pfadangaben in Tabellenreferenzen. Tabellenreferenzen über Pfad oder Dateiname und Erweiterung müssen in einfachen oder doppelten Anführungszeichen eingeschlossen werden. Zum Beispiel:

```
SELECT *
FROM [parts]
```

```
SELECT *
FROM "c:\sample\parts.dat"
```

Falls Sie die Dateierweiterung für den Namen einer Tabelle weglassen, wird automatisch ".dat" angehängt.

1.3.4.2.2 Spaltennamen

Wie der ANSI-Standard-SQL beschränkt TurboDB alle Spaltennamen auf ein einzelnes, aus alphanumerischen Zeichen und dem Unterstrich, "_", bestehendes Wort. Um gleiche Spaltennamen in verschiedenen Tabellen zu unterscheiden können Sie den Tabellennamen vor dem Spaltenbezeichner angeben.

```
SELECT Employee_Id
FROM Employee
```

oder

```
SELECT Employee.Employee_Id
FROM Employee
```

Außerdem kann TurboSQL auch deutsche Umlaute für Tabellen- und Spaltennamen benutzen:

```
SELECT Kürzung
FROM Beiträge
```

Für Spaltennamen, die Leerzeichen oder Spezialzeichen enthalten und um zwischen Spaltennamen und Schlüsselwörtern unterscheiden zu können, kann der Spaltenname in doppelten Anführungszeichen oder eckigen Klammern geschrieben werden.

```
SELECT "Employee Id", [Employee Id]
FROM [Update]
```

1.3.4.2.3 String Literale

Zeichenketten Literale sind von einfachen Anführungszeichen umschlossen. Um ein einzelnes Anführungszeichen innerhalb einer Zeichenkette zu bezeichnen, sind zwei einzufügen. Hier Beispiele für gültige Zeichenketten Literale:

```
'This is a string literal'
```

```
'This is a ''single-quoted'' string literal'
```

```
'This is a "double-quoted" string literal'
```

Ein ungültiges Beispiel:

```
'This isn't a valid string literal'
```

Hinweis

In früheren TurboDB Versionen waren auch doppelte Anführungszeichen für Zeichenketten Literale möglich. Für eine bessere SQL-Kompatibilität werden Doppelt-Anführungsstriche jetzt nur noch für Tabellen- und Spaltenbezeichner verwendet.

1.3.4.2.4 Datumsformate

Datumskonstanten können entweder im proprietären TurboDB Format angegeben werden, das keine Anführungszeichen benötigt, oder in drei verschiedenen standardisierten Formaten. Wo lokale Datumsformate erlaubt sind, sind nur das TurboDB Format und das aktuell eingestellte lokale Format gültig. In diesen Situationen werden die drei Standard-Datumsformate nicht akzeptiert.

Das native Format ist tt.mm.jjjj. Dieses Format entspricht der natürlichen Schreibweise und kann vom Parser nicht mit einem arithmetischen Ausdruck verwechselt werden. Aus diesem Grund, ist es unnötig (sogar verboten) solch ein Datumsliteral in Anführungszeichen zu setzen. Beispiel:

```
SELECT * FROM orders
WHERE saledate <= 31.12.2001
```

sucht nach Verkäufen bis zum 31 Dezember 2001. Dieses Format ist immer gültig und wird auch

immer auf dieselbe Weise interpretiert. Es ist immer dann zu bevorzugen, wenn das Datumsformat nicht an die lokalen Einstellungen des Computers angepasst werden soll.

Die in Anführungszeichen gesetzten Datumsformate sind immer dann gültig, wenn lokale Formate nicht erlaubt sind. Es gibt ein amerikanisches, ein internationales und ein europäisches Datumsformat. Dem in Anführungszeichen gesetzten String ist das Schlüsselwort *DATE* voranzustellen:

```
SELECT * FROM orders
WHERE saledate <= DATE'12/31/2001'
```

oder

```
SELECT * FROM orders
WHERE saledate <= DATE'2001-12-31'
```

Das deutsche Format funktioniert genauso:

```
SELECT * FROM orders
WHERE saledate <= DATE'31.12.2001'
```

Führende Nullen bei Monats- und Tagesfeldern sind optional. Wird beim Jahr nicht das Jahrhundert angegeben, wird das 20te Jahrhundert für die Jahre von 50 bis 99 und das 21te Jahrhundert für die Jahre von 00 bis 49 angenommen.

Das Schlüsselwort *DATE* kann entfallen, wenn der Typ der Zeichenkette wie in den beiden obigen Beispielen offensichtlich ist.

Beispiel

```
SELECT * FROM orders
WHERE (saledate > 1.1.89) AND (saledate <= 31.12.20)
```

sucht nach Verkäufen zwischen dem 1. Januar 1989 und dem 31. Dezember 2020.

1.3.4.2.5 Zeitformate

Zeitkonstanten können entweder im proprietären TurboDB Format, das keine Anführungszeichen benötigt (hh:mm:ss), oder in zwei verschiedenen standardisierten Formaten angegeben werden. Sind lokale Datumseinstellungen erlaubt, ist nur das TurboDB Format und das aktuell eingestellte lokale Format gültig. In diesen Situationen werden die in Anführungszeichen gesetzten Standard-Zeitformate nicht akzeptiert.

Das native Format erwartet Zeit-Literale in der Form hh:mm wobei hh für die Stunden und mm für die Minuten steht. Turbo SQL benutzt die 24 Stunden Skala, das bedeutet 2:10 ist früh am morgen (2:10 AM) während 14:10 am frühen Nachmittag ist (2:10 PM).Der Zeitausdruck darf nicht in Anführungszeichen stehen.

```
INSERT INTO WorkOrder
(ID, StartTime) VALUES ('B00120', 22:30)
```

Dieses Format ist immer gültig und wird immer auf dieselbe Weise interpretiert. Es ist immer dann zu bevorzugen, wenn das Zeitformat nicht an die lokalen Einstellungen des Computers angepasst werden soll.

Falls Sie es vorziehen, können Sie die Zeit auch im amerikanischen *hh.mm am/pm* Format angeben. Dazu müssen Sie das Literal mit Anführungszeichen setzen und das Schlüsselwort *TIME* voranstellen:

```
INSERT INTO WorkOrder
(ID, StartTime) VALUES ('B00120', TIME'10.30 pm')
```

Ist der Typ der Zeichenkette offensichtlich, wie im vorangegangenen Beispiel, kann das Schlüsselwort *TIME* weggelassen werden. Im folgenden Beispiel dagegen muss *TIME* stehen:

```
SELECT StartTime - TIME'12:00:00 pm' FROM WorkOrder
```

Hinweis

Die erste Alternative, das native Format ohne Anführungszeichen, kann nicht mit der

VCL-Komponente *TTdbQuery* verwendet werden. Der VCL-Parser für SQL interpretiert den Doppelpunkt als Start-Zeichen eines Parameters. Sie können diesen entweder löschen oder einfach ignorieren, das Statement wird in jedem Fall korrekt ausgeführt.

1.3.4.2.6 Zeitstempel Format

Zeitstempel-Formate (TimeStamp oder DateTime) können entweder im proprietären TurboDB Format angegeben werden, das keine Anführungszeichen benötigt (*dd.mm.yyyy_hh:mm:ss*), oder in drei verschiedenen standardisierten Formaten. Wo lokale Datumsformate erlaubt sind, sind nur das TurboDB Format und das aktuell eingestellte lokale Format gültig. In diesen Situationen werden die drei Standard-Datumsformate nicht akzeptiert.

Das native Format für DateTime Literale setzt sich aus einem Datums- und Zeitanteil zusammen, getrennt duch einen Unterstrich. Anführungszeichen können optional gesetzt werden.

```
SELECT * FROM WorkOrder
WHERE StartTime >= 31.1.2001_14:10:00
```

oder

```
SELECT *
FROM WorkOrder
WHERE StartTime >= '31.1.2001_14:10:00'
```

Dieses Format ist immer gültig und wird auch immer auf dieselbe Weise interpretiert. Es ist immer dann zu bevorzugen, wenn das TimeStamp-Format nicht an die lokalen Einstellungen des Computers angepasst werden soll.

Eine andere Möglichkeit ist es ein DateTime in einfache Anführungszeichen zu setzen und das Schlüsselwort TIMESTAMP voranzustellen. Auch hier gibt es drei verschiedene Möglichkeiten der Darstellung:

Das amerikanische Format

```
SELECT * FROM WorkOrder
WHERE StartTime >= TIMESTAMP'1/31/2001 2:10:00 pm'
```

Das internationale Format:

```
SELECT * FROM WorkOrder
WHERE StartTime >= TIMESTAMP'2001-1-31 14:10:00'
```

Das europäische Format:

```
SELECT * FROM WorkOrder
WHERE StartTime >= TIMESTAMP'31.1.2001 14:10:00'
```

Wenn, wie in den obigen Beispielen, die Natur der DateTime-Zeichenkette eindeutig ist, kann das Schlüsselwort TIMESTAMP weggelassen werden.

Hinweis
Die erste Variante mit dem nativen Format ohne umschließende Klammer kann mit der Delphi-Komponente TTdbQuery nicht benutzt werden. Der VCL-Parser für SQL-Befehle interpretiert den Doppelpunkt als Startzeichen eines Parameters. Sie können diesen entweder löschen oder einfach ignorieren, das Statement wird in jedem Fall korrekt ausgeführt.

1.3.4.2.7 Boolsche Konstanten

Die Boolschen Konstantenwerte TRUE und FALSE müssen ohne Anführungszeichen angegeben werden. Eine Unterscheidung zwischen Groß- und Kleinschreibung wird nicht vorgenommen.

```
SELECT *
FROM transfers
WHERE (paid = 'True') AND NOT (incomplete = FALSE)
```

1.3.4.2.8 Tabellenkorrelationsnamen

Tabellenkorrelationsnamen werden verwendet, um eine Spalte explizit mit der Tabelle zu verknüpfen, aus der sie stammt. Dies ist besonders nützlich, wenn mehrere Spalten gleichen Namens in derselben Anfrage erscheinen, üblicherweise in Mehrfach-Tabellenabfragen. Ein Tabellenkorrelationsname wird definiert, indem der Tabellenreferenz in FROM-Klausel einer SELECT-Abfrage ein eindeutiger Bezeichner nachgestellt wird. Dieser Bezeichner oder Tabellenkorrelationsname kann dann verwendet werden, um einem Spaltennamen vorangestellt zu werden.

Ist der Tabellenname kein String in Anführungszeichen, so ist der Tabellenname der implizit vorgegebene Korrelationsname. Ein expliziter Korrelationsname, welcher mit dem Tabellennamen übereinstimmt, muß in der FROM-Klausel nicht angegeben werden, und der Tabellenname kann Spaltennamen in anderen Teilen der Anweisung vorangestellt werden.

```
SELECT *
FROM "/home/data/transfers.dat" transfers
WHERE transfers.incomplete = False
```

1.3.4.2.9 Spaltenkorrelationsnamen

Verwenden Sie das Schlüsselwort AS, um einer Spalte, einem aggregierten Wert oder einer Konstante einen Korrelationsnamen zuzuweisen. In der folgenden Anweisung sind die Token Sub und Word Spaltenkorrelationsnamen.

```
SELECT SUBSTRING(company FROM 1 FOR 1) AS sub, Text word
FROM customer
```

1.3.4.2.10 Parameter

Befehls-Parameter werden durch einen Doppelpunkt eingeleitet:

```
INSERT INTO Customer (Name) VALUES(:Name)
```

Der Name des Parameters ist der Bezeichner ohne den Doppelpunkt, d.h. Name im obigen Fall. Um auf den Parameter in einer API-Funktion oder von einer Komponenten-Bibliothek aus zuzugreifen, wird der Bezeichner ohne Doppelpunkt benutzt.

Bei der Arbeit mit dem ODBC Interface, werden auch unbenannte Parameter unterstützt:

```
INSERT INTO Customers (Name) VALUES (?)
```

1.3.4.2.11 Eingebettete Kommentare

Es gibt zwei Möglichkeiten Text als Kommentar in ein SQL Statement einzubetten, die vergelichbar mit den Konventionen unter C++ sind. Entweder Sie benutzen /* und */ um auch mehrzeiligen Kommentar zu umschließen oder Sie beginn den Kommentar mit //, das Ende wird durch das Ende der Zeile bestimmt.

Beispiel:

```
/* This finds all requests that have come in in the period of time we
are looking at. */
SELECT * FROM Request
// Requests from the time before the Euro came
WHERE Date < '1/1/2002'
```

1.3.4.3 Systemtabellen

TurboDB verwendet Systemtabellen um Management Informationen der Datenbank zu speichern und das Informations-Schema dem Benutzer anzubieten. Die im Folgenden aufgeführten Tabellen korrespondieren zu ihren Entsprechungen in SQL 92

Es hängt vom Management Level der Datenbank ab, ob diese Tabellen permanent oder temporär

vorhanden sind. In beiden Fällen können sie abgefragt werden.

sys_UserTables	Liste der Benutzer-Tabellen der Datenbank.
sys_UserColumns	Liste der sichtbaren Spalten der Benutzer-Tabellen.
sys_UserTableConstraints	Liste der Namen aller Schlüssel, Gültigkeitsbedingungen und Fremdschlüssel aller Benutzer-Tabellen.
sys_UserKeyColumns	Liste aller Spalten von Benutzer-Tabellen die Bestandteil eines (Primär-, Kandidat- oder Fremd-) Schlüssels.
sys_UserCheckConstraints	Lists the check constraints including the check condition.
sys_UserReferentialConstraints	Indicates the referenced unique constraint and the referential action for each foreign key in the database.
sys_UserIndexes	Lists the indexes of all user tables.
sys_UserIndexColumns	Lists all indexed columns of all user tables.
sys_UserRoutines	Lists all stored procedures of the database.

Beispiele:

```
select ColumnName, DataType from sys_UserColumns where TableName =
'TableA'
```

Stellt die Spalten und deren Datentypen für die Tabelle *TableA dar*.

```
select I.TableName, I.IndexName, C.ColumnName
from sys_UserIndexes I join sys_UserIndexColumns C on I.TableName = C.
TableName and I.IndexName = C.IndexName
where I.IsUnique = True
```

Liefert die Spalten aller eindeutigen Indexe.

1.3.4.4 Data Manipulation Language

Turbo SQL unterstützt folgende Data Manipulation Language- (DML-) Anweisungen, Klauseln, Funktionen und Prädikate:

Statements

DELETE	Löscht bestehende Datensätze aus einer Tabelle.
INSERT	Fügt einer Tabelle neue Daten hinzu
SELECT	Liest bestehende Daten aus einer Tabelle aus.
UPDATE	Verändert bestehende Daten in einer Tabelle..

Klauseln

FROM	Gibt die für die Anweisung verwendeten Tabellen an.
GROUP BY	Gibt die Spalten an, die zum Gruppieren von Zeilen verwendet werden.
HAVING	Gibt Filterkriterien unter Verwendung aggregierter Daten an.
ORDER BY	Gibt die Spalten an, anhand deren die Ergebnismenge zu sortieren ist.
WHERE	Gibt Filterkriterien zum Begrenzen der abgerufenen Zeilen an.
Unterabfragen	Vergleich mit dem Ergebnis einer Abfrage mit IN, ANY, SOME, ALL und EXISTS.

Funktionen, Prädikate und Operatoren

Allgemeine	Berechnungen und Vergleiche mit Zahlen, Zeichenketten, Datum- und

Funktionen und Operatoren	Zeitwerten etc.
Arithmetische Funktionen und Operatoren	Berechnungen mit Zahlen
Zeichenketten Funktionen und Operatoren	Berechnungen mit Zeichenketten
Datum und Zeit Funktionen und Operatoren	Berechnungen mit Datum- und Zeitwerten
Aggregat Funktionen	Statistik
Sonstige Funktionen und Operatoren	Berechnungen, die sich nicht in eine der Kategorien einordnen lassen
Tabellen Operatoren	Kombiniert zwei Tabellen zu einer neuen
Unterabfragen	Vergleicht Datensätze mit der Ergebnismenge einer anderen Abfrage
Volltextsuche	Ausführen einer Volltextsuche

1.3.4.4.1 DELETE Anweisung

Löscht eine oder mehrere Zeilen aus einer Tabelle.

```
DELETE FROM table_reference
[WHERE predicates]
```

Beschreibung

Verwenden Sie DELETE, um eine oder mehrere Zeilen aus einer Tabelle zu löschen.

```
DELETE FROM [employee]
```

Die optionale WHERE-Klausel beschränkt das Löschen von Zeilen auf einen Teil der Zeilen in der Tabelle. Ist keine WHERE-Klausel angegeben, so werden alle Zeilen in der Tabelle gelöscht:

```
DELETE FROM [employee]
WHERE empno > 2300
```

Die Tabellenreferenz kann der DELETE-Anweisung nicht als Parameter übergeben werden.

Wichtiger Hinweis:

Das DELETE-Statement ohne WHERE-Klausel löscht alle Zeilen eine Tabelle ohne Einschränkungen wie z.B. Fremdschlüssel zu prüfen. Dies ist ein Leistungsmerkmal zum schnellen Leeren von Tabellen.

1.3.4.4.2 FROM Klausel

Gibt die Tabellen an, aus denen eine SELECT-Anweisung Daten entnimmt.

```
FROM table_reference [, table_reference...]
```

Beschreibung

Verwenden Sie eine FROM-Klausel, um die Tabellen anzugeben, aus denen eine SELECT-Anweisung Daten entnimmt. Der Wert für eine FROM-Klausel ist eine durch Kommas getrennte Liste von Tabellennamen. Angegebene Tabellennamen müssen den Benennungskonventionen von Turbo SQL entsprechen. Beispielsweise entnimmt die folgende SELECT-Anweisung Daten aus einer einzelnen TurboDB-Tabelle:

```
SELECT * FROM [customer]
```

```
SELECT * FROM
customer, orders
```

```
SELECT * FROM
customer JOIN orders ON orders.CustNo = customer.CustNo
```

Anwendung

SELECT

1.3.4.4.3 GROUP BY Klausel

Verbindet Zeilen mit gemeinsamen Spaltenwerten in einzelnen Zeilen.

```
GROUP BY column_reference [, column reference...]
```

Beschreibung

Verwenden Sie eine GROUP BY-Klausel zum Verbinden von Zeilen mit denselben Spaltenwerten in einer einzigen Zeile. Die Kriterien zum Verbinden von Zeilen basieren auf den Werten in denjenigen Spalten, die in der GROUP BY-Klausel angegeben sind. Der Zweck der Verwendung einer GROUP BY-Klausel besteht darin, einen oder mehrere Spaltenwerte in einem einzelnen Wert zu verbinden (aggregieren) und eine oder mehrere Spalten zum eindeutigen Identifizieren der aggregierten Werte bereitzustellen. Eine GROUP BY-Klausel kann nur verwendet werden, wenn auf eine oder mehrere Funktionen eine Aggregatfunktion angewendet wird.

Der Wert für die GROUP BY-Klausel ist eine durch Kommas getrennte Spaltenliste. Jede Spalte in dieser Liste muß folgenden Kriterien entsprechen:

- Sie muß in einer der in der FROM-Klausel der Abfrage angegebenen Tabellen stehen.
- Sie muß in der SELECT-Klausel der Abfrage stehen.
- Auf sie darf keine Aggregatfunktion angewendet werden.

Wird eine GROUP BY-Klausel verwendet, so müssen alle Tabellenspalten in der SELECT-Klausel der Abfrage mindestens einem der folgenden Kriterien genügen, oder sie darf nicht in der SELECT-Klausel stehen:

- Sie muß in der GROUP BY-Klausel der Abfrage stehen.
- Sie muß im Subjekt einer Aggregatfunktion stehen.

Für Literale in der SELECT-Klausel gelten die genannten Kriterien nicht.

Die Unterscheidbarkeit von Zeilen basiert auf den Spalten in der angegebenen Spaltenliste. Alle Zeilen mit gleichen Werten in diesen Spalten werden in einer einzigen Zeile (oder logischen Gruppe) kombiniert. Für Spalten, die Subjekt einer Aggregatfunktion sind, werden die Werte über alle Zeilen der Gruppe kombiniert. Alle Spalten, die nicht Subjekt einer Aggregatfunktion sind, behalten ihre Werte bei und dienen dazu, die Gruppe eindeutig zu identifizieren. So werden beispielsweise in der folgenden SELECT-Anweisung die Werte in der Spalte SALES basierend auf unterschiedlichen Werten in der Spalte COMPANY in Gruppen aggregiert (summiert). Auf diese Weise werden Verkaufssummen für jede Firma berechnet:

```
SELECT company, SUM(sales) AS TOTALSALES
FROM sales1998
GROUP BY company
ORDER BY company
```

Eine Spalte kann in einer GROUP BY-Klausel durch einen Spaltenkorrelationsnamen anstelle tatsächlicher Spaltennamen referenziert werden. Die nachfolgende Anweisung gruppiert unter Verwendung der ersten Spalte, COMPANY, repräsentiert durch den Spaltenkorrelationsnamen Co:

```
SELECT company AS Co, SUM(sales) AS TOTALSALES
FROM sales1998
GROUP BY Co
ORDER BY 1
```

Hinweis

- Abgeleitete Werte (berechnete Felder) können nicht als Basis für eine GROUP BY-Klausel verwendet werden.
- Spaltenreferenzen können einer GROUP BY-Klausel nicht als Parameter übergeben werden.

Anwendung

SELECT, wenn Aggregatfunktionen verwendet werden

1.3.4.4.4 HAVING Klausel

Gibt Filterbedingungen für eine SELECT-Anweisung an.

```
HAVING predicates
```

Beschreibung

Verwenden Sie eine HAVING-Klausel, um die von einer SELECT-Anweisung abgerufenen Zeilen auf eine Untermenge von Zeilen zu beschränken, für welche die aggregierten Spaltenwerte den angegebenen Kriterien genügen. Eine HAVING-Klausel kann nur dann in einer SELECT-Anweisung verwendet werden, wenn:

- die Anweisung auch eine GROUP BY-Klausel enthält,
- eine oder mehrere Spalten Subjekte von Aggregatfunktionen sind.

Der Wert für eine HAVING-Klausel sind ein oder mehrere logische Ausdrücke oder Prädikate, die für jede aggregierte, der Tabelle entnommene Zeile TRUE oder FALSE ergeben. Nur jene Zeilen, für welche die Prädikate TRUE ergeben, werden von einer SELECT-Anweisung entnommen. So entnimmt beispielsweise folgende SELECT-Anweisung alle Zeilen, in denen die Verkaufssumme für individuelle Verkaufssummen den Wert 1000 übersteigt:

```
SELECT company, SUM(sales) AS TOTALSALES
FROM sales1998
GROUP BY company
HAVING (SUM(sales) >= 1000)
ORDER BY company
```

Mehrere Prädikate müssen durch einen der logischen Operatoren OR oder AND getrennt werden. Jedes Prädikat kann durch den Operator NOT negiert werden. Klammern können zur Erzeugung verschiedener Zeilenevaluierungskriterien verwendet werden, um logische Vergleiche und Gruppen von Vergleichen zu isolieren.

Eine SELECT-Anweisung kann sowohl eine WHERE- als auch eine HAVING-Klausel enthalten. Die WHERE-Klausel filtert die zu aggregierenden Daten unter Verwendung der Spalten, die nicht Subjekte von Aggregatfunktionen sind. Die HAVING-Klausel filtert dann die Daten nach der Aggregation unter Verwendung derjenigen Spalten, die Subjekte von Aggregatfunktionen sind. Folgende SELECT-Abfrage führt die gleiche Operation wie obige durch, die Daten sind jedoch auf jene beschränkt, deren Spalte STATE den Wert "CA" hat:

```
SELECT company, SUM(sales) AS TOTALSALES
FROM sales1998
WHERE (state = 'CA')
GROUP BY company
HAVING (SUM(sales) >= 1000)
ORDER BY company
```

Hinweis

Eine HAVING-Klausel filtert Daten nach der Aggregation einer GROUP BY-Klausel. Verwenden Sie eine WHERE-Klausel zum Filtern basierend auf Zeilenwerten vor der Aggregation.

Anwendung

SELECT mit GROUP BY

1.3.4.4.5 INSERT Anweisung

Fügt einer Tabelle eine oder mehrere Datenzeilen hinzu.

```
INSERT INTO table_reference
[(columns_list)]
VALUES (update_atoms)
```

Beschreibung

Verwenden Sie die Anweisung INSERT, um einer Tabelle eine oder mehrere Datenzeilen hinzuzufügen.

Verwenden Sie eine Tabellenreferenz in der INTO-Klausel, um die Tabelle anzugeben, welche die eingehenden Daten aufnehmen soll.

Bei der Spaltenliste handelt es sich um eine durch Kommas getrennte, in Klammern eingeschlossene, optionale Liste der Spalten in der Tabelle. Die VALUES-Klausel ist eine durch Kommas getrennte, in Klammern eingeschlossenen Liste von Aktualisierungsatomen. Ist keine Spaltenliste angegeben, so werden eingehende Aktualisierungswerte (Aktualisierungsatome) in den Feldern gespeichert, wie sie sequentiell in der Tabellenstruktur definiert sind. Aktualisierungsatome werden auf die Spalten in der Reihenfolge angewendet, in der die Aktualisierungsatome in der VALUES-Klausel aufgeführt sind. Außerdem müssen so viele Aktualisierungsatome vorhanden sein wie Spalten in der Tabelle.

```
INSERT INTO [holdings]
VALUES (4094095, "BORL", 5000, 10.500, 2.1.1998)
```

Ist eine explizite Spaltenliste angegeben, so werden eingehende Aktualisierungsatome (in der Reihenfolge, in der sie in der VALUES-Klausel auftreten) in den aufgeführten Spalten (in der Reihenfolge, in der sie in der Spaltenliste auftreten) gespeichert. In etwaigen Spalten, die nicht in der Spaltenliste stehen, werden NULL-Werte gespeichert:

```
INSERT INTO [customer]
(custno, company)
VALUES (9842, "dataWeb GmbH")
```

Um Tabellen Zeilen aus einer anderen Tabelle hinzuzufügen, lassen Sie das Schlüsselwort VALUES weg und verwenden Sie eine Unterabfrage als Quelle der neuen Zeilen:

```
INSERT INTO [customer]
(custno, company)
SELECT custno, company
FROM [oldcustomer]
```

1.3.4.4.6 ORDER BY Klausel

Sortiert die von einer SELECT-Anweisung abgerufenen Werte.

```
ORDER BY column_reference [, column_reference...] [ASC|DESC]
```

Beschreibung

Verwenden Sie die Klausel ORDER BY, um die von einer SELECT-Anweisung abgerufenen Zeilen basierend auf den Werten einer oder mehrerer Spalten zu sortieren.

Bei dem Wert für die Klausel ORDER BY handelt es sich um eine durch Kommas getrennte Liste von Spaltennamen. Die Spalten in dieser Liste müssen auch in der SELECT-Klausel der Abfrageanweisung stehen. Die Spalten in der ORDER BY-Liste können aus einer oder mehreren Tabellen kommen. Eine Zahl, welche die relative Position einer Spalte in der SELECT-Klausel repräsentiert, kann anstelle eines Spaltennamens verwendet werden. Spaltenkorrelationsnamen können ebenfalls in der Spaltenliste einer ORDER BY-Klausel verwendet werden.

Verwenden Sie ASC (bzw. ASCENDING), um festzulegen, daß die Sortierung in aufsteigender Reihenfolge erfolgen soll (vom Kleinsten zum Größten Wert). Verwenden Sie DESC (bzw. DESCENDING) für eine absteigende Sortierungsreihenfolge (vom Größten zum Kleinsten Wert). Wenn nicht angegeben, wird ASC als Vorgabe verwendet.

Die folgende Anweisung sortiert die Ergebnismenge aufsteigend nach dem der Spalte

LASTINVOICEDATE entnommenen Jahr, dann absteigend nach der Spalte STATE und dann aufsteigend nach der Spalte COMPANY, umgewandelt in Großbuchstaben:

```
SELECT EXTRACT(YEAR FROM lastinvoicedate) AS YY, state, UPPER(company)
FROM customer
ORDER BY YY DESC, state ASC, 3
```

Spaltenreferenzen können einer ORDER BY-Klausel nicht als Parameter übergeben werden.

Anwendung

SELECT

1.3.4.4.7 SELECT Anweisung

Liest Daten aus Tabellen aus.

```
SELECT [TOP number] [DISTINCT] * | column_list
FROM table_reference
[WHERE predicates]
[ORDER BY order_list]
[GROUP BY group_list]
[HAVING having_condition]
```

Beschreibung

Verwenden Sie die Anweisung SELECT, um

- aus einer Tabelle eine einzelne Zeile oder einen Teil einer Zeile abzurufen. Dies wird als Singleton-Select bezeichnet.
- aus einer Tabelle mehrere Zeilen oder Teile von Zeilen abzurufen.
- aus der Verbindung zweier oder mehrerer Tabellen verwandte Zeilen oder Teile von Zeilen abzurufen.

Die SELECT-Klausel definiert die Liste von Elementen, die von der SELECT-Anweisung zurückgegeben wird. Die SELECT-Klausel verwendet eine durch Kommas getrennte Liste, die sich aus folgenden Bestandteilen zusammensetzt: Spalten von Tabellen, Konstanten und durch Funktionen veränderte Spalten- oder Konstantenwerte. Konstantenwerte in der Spaltenliste können der SELECT-Anweisung als Parameter übergeben werden. Sie können Parameter nicht verwenden, um Spaltennamen zu repräsentieren. Verwenden Sie das Sternzeichen ("*"), um Werte aus allen Spalten auszulesen.

Die Spalten in der Spaltenliste für die SELECT-Klausel können aus einer oder mehreren Tabellen stammen, sofern diese Tabellen in der FROM-Klausel aufgeführt sind. Die FROM-Klausel identifiziert die Tabelle(n), aus der/denen die Daten ausgelesen werden.

Falls das Schlüsselwort TOP angegeben ist, wird die Anzahl der Datensätze der Ergebnismenge auf die gegebenen Zahl eingeschränkt. Top wird nach allen anderen Klauseln ausgewertet und beachtet daher die gewählte Sortierung oder Gruppierung, falls ORDER BY und/oder GROUP BY angegeben wurden.

Wenn das Schlüsselwort DISTINCT angegeben ist, werden doppelte Zeilen in der Ergebnis-Tabelle unterdrückt. DISTINCT kann nicht zusammen mit GROUP BY verwendet werden. Falls eine SELECT-Anweisung sowohl GROUP BY als auch DISTINCT enthält, wird das Schlüsselwort DISTINCT ignoriert.

Die folgende Anweisung entnimmt Daten für zwei Spalten aus allen Zeilen einer Tabelle:

```
SELECT custno, company
FROM orders
```

Siehe auch

JOIN, UNION, INTERSECT, EXCEPT

1.3.4.4.8 UPDATE Anweisung

Ändert eine oder mehrere bestehende Zeilen in einer Tabelle.

```
UPDATE table_reference
SET column_ref = update_value [, column_ref = update_value...]
[WHERE predicates]
```

Beschreibung

Verwenden Sie die Anweisung UPDATE, um einen oder mehrere Spaltenwerte in einer oder mehreren bestehenden Zeilen einer Tabelle zu ändern.

Verwenden Sie eine Tabellenreferenz in der UPDATE-Klausel, um die Tabelle anzugeben, welche die Änderungen erhalten soll.

Die SET-Klausel ist eine durch Kommas getrennte Liste von Aktualisierungsausdrücken. Jeder Ausdruck setzt sich aus dem Namen einer Spalte, dem Zuweisungsoperator (=) und dem Aktualisierungswert (Aktualisierungsatom) für diese Spalte zusammen.

```
UPDATE salesinfo
SET taxrate = 0.0825
WHERE (state = 'CA')
```

Die optionale WHERE-Klausel beschränkt Aktualisierungen auf einen Teil der Zeilen in der Tabelle. Ist keine WHERE-Klausel angegeben, so werden alle Zeilen in der Tabelle unter Verwendung der Aktualisierungsausdrücke der SET-Klausel aktualisiert.

Bei Verwendung einer Unterabfrage können die Werte auch einer anderen Tabelle stammen:

```
UPDATE salesinfo
SET taxrate = (SELECT newesttaxrate FROM [globals])
```

Siehe auch

INSERT, DELETE

1.3.4.4.9 WHERE Klausel

Gibt Filterbedingungen für eine SELECT- oder UPDATE-Anweisung an.

```
WHERE predicates
```

Beschreibung

Verwenden Sie eine WHERE-Klausel, um die Auswirkungen einer SELECT- oder UPDATE-Anweisung auf eine Untermenge von Zeilen in der Tabelle zu beschränken. Die Verwendung einer WHERE-Klausel ist optional.

Der Wert für eine WHERE-Klausel ist ein oder mehrere logische Ausdrücke oder Prädikate, die für jede Zeile in der Tabelle TRUE oder FALSE ergeben. Nur jene Zeilen, für welche die Prädikate TRUE ergeben, werden von einer SELECT-Anweisung entnommen oder von einer UPDATE-Anweisung aktualisiert. Beispielsweise entnimmt die folgende SELECT-Anweisung alle Zeilen, für welche die Spalte STATE den Wert "CA" enthält:

```
SELECT company, state
FROM customer
WHERE state = 'CA'
```

Mehrere Prädikate müssen durch einen der logischen Operatoren OR oder AND getrennt werden. Jedes Prädikat kann durch den Operator NOT negiert werden. Klammern können verwendet werden, um logische Vergleiche und Gruppen von Vergleichen zu isolieren. Dadurch werden unterschiedliche Zeilenevaluierungskriterien produziert. So entnimmt beispielsweise die folgende SELECT-Anweisung alle Zeilen, für welche die Spalte STATE den Wert "CA" enthält, sowie diejenigen mit dem Wert "HI":

```
SELECT company, state
FROM customer
WHERE (state = 'CA') OR (state = 'HI')
```

Die folgende SELECT-Anweisung entnimmt alle Zeilen, in denen die Spalte SHAPE die Werte

"round" oder "square" und die Spalte COLOR den Wert „red" enthält. Eine Zeile, die beispielsweise in der Spalte SHAPE der Wert „round" und in der Spalte COLOR der Wert „blue" enthält, würde nicht entnommen werden.

```
SELECT shape, color, cost
FROM objects
WHERE ((shape = 'round') OR (shape = 'square')) AND (color = 'red')
```

Ohne die Klammern, durch die die Rangfolge der logischen Operatoren verändert wird, erhalten Sie ein völlig anderes Ergebnis (siehe die folgende Anweisung). Diese Anweisung entnimmt die Zeilen, die in der Spalte SHAPE "round" enthalten, unabhängig vom in der Spalte COLOR vorhandenen Wert. Desweiteren werden Zeilen entnommen, die in der Spalte SHAPE den Wert "square" und in der Spalte COLOR den Wert "red" enthalten. Im Gegensatz zum obigen Beispiel würden hier Zeilen entnommen, die in SHAPE "round" und in COLOR "blue" enthalten.

```
SELECT shape, color, cost
FROM objects
WHERE shape = 'round' OR shape = 'square' AND color = 'red'
```

Hinweis

Eine WHERE-Klausel filtert Daten vor der Gruppierung einer GROUP BY-Klausel. Verwenden Sie zum Filtern basierend auf aggregierten Werten eine HAVING-Klausel.

Anwendung

SELECT, UPDATE, DELETE

1.3.4.4.10 Allgemeine Funktionen und Operatoren

Hier ist eine Liste von Funktionen und Operatoren, die in *TurboSQL* Ausdrücken verwendet werden können. Die Liste setzt sich aus wenigen Standard SQL Funktion und vielen zusätzlichen *TurboDB* Funktionen zusammen.

=

Syntax

```
expr1 = expr2
```

Beschreibung

Test auf Gleichheit.

<

Syntax

```
expr1 < expr2
```

Beschreibung

Test ob der Ausdruck *expr1* kleiner als *expr2* ist.

<=

Syntax

```
expr1 <= expr2
```

Beschreibung

Test ob der Ausdruck expression *expr1* kleiner oder gleich *expr2* ist.

>

Syntax

```
expr1 > expr2
```

Beschreibung

Test ob der Ausdruck *expr1* größer als *expr2* ist.

>=

Syntax

```
expr1 >= expr2
```

Beschreibung

Test ob der Ausdruck *expr1* größer oder gleich *expr2* ist.

BETWEEN ... AND ...

Syntax

```
expr1 BETWEEN expr2 AND expr3
```

Beschreibung

Test ob der Ausdruck *expr1* größer oder gleich *expr2* und kleiner oder gleich *expr3* ist.

IN

Syntax

```
expr IN (expr1, expr2, expr3, ...)
```

Beschreibung

Test ob *expr* mit einem der Ausdrücke *expr1*, *expr2*, *expr3*, ... übereinstimmt.

AND

Syntax

```
cond1 AND cond2
```

Beschreibung

Test ob sowohl *cond1* als auch *cond2* wahr ist.

OR

Syntax

```
cond1 OR cond2
```

Beschreibung

Test ob *cond1* oder *cond2* wahr ist.

NOT

Syntax

```
NOT cond
```

Beschreibung

Test ob *cond* falsch ist.

CASE

Syntax

```
CASE
  WHEN cond1 THEN expr1
  WHEN cond2 THEN expr2
  ...
  [ELSE exprN]
END
```

```
CASE expr
  WHEN exprA1 THEN exprB1
  WHEN exprA2 THEN exprB2
  ...
  [ELSE exprBN]
```

```
END
```

Beschreibung

Die erste Form der case Operation ermittelt den ersten Ausdruck für den die Bedingung wahr ist. Die zweite liefert den B Ausdruck, dessen A Ausdruck identisch ist zu *expr*.

Beispiele

```
CASE WHEN Age < 8 THEN 'infant'WHEN Age < 18 THEN 'teenager' WHEN Age <
30 THEN 'twen' ELSE 'adult' END
CASE Status WHEN 0 THEN 'OK' WHEN 1 THEN 'WARNING' WHEN 2 THEN 'ERROR'
END
```

CAST

Syntax

```
CAST(value AS type [COLLATE collation])
```

Beschreibung

Wandelt *value* falls möglich in den angegebenen Datentyp *type* um. Die Cast Operation kann Zeichenketten abschneiden und die Genauigkeit von Zahlen vermindern. Falls eine Umwandlung nicht möglich ist, führt CAST zu einem Fehler.
Beim Cast auf einen String-Typ kann zusätzlich ein angepasstes Sortierschema angegeben werden.

Beispiele

```
CAST(time AS CHAR(10)) --Converts the time in its string representation
CAST(time AS CHAR(3)) --Displays only the first three characters
CAST(username AS CHAR(50) Collate German_cs_as) --Sets a custom sort
collation on field username
CAST(amount AS INTEGER)) --Looses the digits after the decimal point
CAST('abc' AS BIGINT) --Raises a conversion error
CAST(34515 AS BYTE) --Raises an overflow error
```

Siehe auch

Allgemeine Funktionen und Operatoren
Arithmetische Funktionen und Operatoren
String Funktionen und Operatoren
Datum und Zeit Funktionen und Operatoren
Aggregat Funktionen
Sonstige Funktionen und Operatoren
TurboPL Functionen und Operatoren

1.3.4.4.11 Arithmetische Funktionen und Operatoren

Dies ist eine Liste arithmetischer Funktionen und Operatoren, die in TurboSQL verwendet werden können.

+

Syntax

```
value1 + value2
```

Beschreibung

Berechnet die Summe beider Werte.

-

Syntax

```
value1 - value2
```

Beschreibung

Berechnet die Differenz beider Werte.

*

Syntax

```
value1 * value2
```

Beschreibung

berechnet das Produkt beider Werte.

/

Syntax

```
value1 / value2
```

Beschreibung

Berechnet den Quotienten aus beiden Werten.

%

Syntax

```
value1 % value2
```

Beschreibung

Berechnet den Modulo-Operator zweier integraler Zahlen.

Kompatibilität

Nur in TurboDB Managed verfügbar.

ARCTAN

Syntax

```
ARCTAN(value)
```

Beschreibung

Berechnet den Arcus Tangens von value

CEILING

Syntax

```
CEILING(value)
```

Beschreibung

Berechnet die kleinste ganze Zahl größer oder gleich dem gegebenen Wert.

Beispiel

```
CEILING(-3.8) --liefert -3.0
CEILING(3.8) --liefert 4.0
```

COS

Syntax

```
COS(value)
```

Beschreibung

Berechnet den Cosinus von value

DIV

Syntax

```
a div b
```

Beschreibung

Ganzzahlige Division

Beispiel

```
35 div 6 --returns 5
-35 div 6 --returns -5
35 div -6 --returns -5
-35 div -6 --returns 5
```

EXP

Syntax

```
EXP(:X DOUBLE) RETURNS DOUBLE
```

Beschreibung

Berechnet das Exponential von value (zur Basis e)

FLOOR

Syntax

```
FLOOR(:X DOUBLE) RETURNS DOUBLE
```

Beschreibung

Berechnet den größten ganzzahligen Wert kleiner oder gleich dem gegebenen Wert.

Beispiel

```
FLOOR(-3.8) --liefert -4.0
FLOOR(3.8) --liefert 3.0
```

FRAC

Syntax

```
FRAC(:X DOUBLE) RETURNS DOUBLE
```

Beschreibung

Errechnet das Bruchteil der realen Zahl.

Beispiel

```
FRAC(-3.8) --returns -0.8
FRAC(3.8) --returns 0.8
```

INT

Syntax

```
INT(:X DOUBLE) RETURNS BIGINT
```

Beschreibung

Errechnet den ganzzahligen Bestandteil einer realen Zahl als ganze Zahl

Beispiel

```
INT(-3.8) --returns -3
INT(3.8) --returns 3
```

LOG

Syntax

```
LOG(:X DOUBLE) RETURNS BIGINT
```

Beschreibung

Errechnet den natürlichen Logarithmus einer realen Zahl.

MOD

Syntax

```
a mod b
```

Beschreibung

Errechnet den Rest einer ganzzahligen Division. Es gilt *a mod b = a - (a div b) * b*.

Beispiel

```
35 mod 6 --returns 5
35 mod -6 --returns 5
-35 mod 6 --returns 5
-35 mod -6 --returns 5
```

ROUND

Syntax

```
ROUND(:X DOUBLE [, :Precision BYTE]) RETURNS DOUBLE
```

Beschreibung

Rundet *value* zu der gegebenen Anzahl an Stellen.

Kompatibilität

Nur in TurboDB Win32 verfügbar.

Beispiel

```
ROUND(3.141592, 3) --liefert 3.142
```

SIN

Syntax

```
SIN(:X DOUBLE) RETURNS DOUBLE
```

Beschreibung

Berechnet den Sinus

SQRT

Syntax

```
SQRT(:X DOUBLE) RETURNS DOUBLE
```

Beschreibung

Berechnet die Quadratwurzel

See also

General Functions and Operators
Arithmetic Functions and Operators
String Functions and Operators
Date and Time Functions and Operators
Aggregation Functions
Miscellaneous Functions and Operators

1.3.4.4.12 Zeichenketten Funktionen und Operatoren

Dies ist eine Liste von Funktionen und Operatoren zur Bearbeitung von Zeichenketten, die in TurboSQL eingesetzt werden können.

||

Syntax

```
string1 || string2
```

Beschreibung

Verknüpft die beiden Zeichenketten.

ASCII

Syntax

```
ASCII(string)
```

Beschreibung

Berechnet den Code des ersten Zeichens in der Zeichenkette. Liefert NULL falls das Argument NULL oder leer ist.

CHAR_LENGTH

Syntax

```
CHAR_LENGTH(string)
```

Beschreibung

Berechnet die Anzahl der Zeichen in der Zeichenkette.

HEXSTR

Syntax

```
HEXSTR(number, width)
```

Beschreibung

Berechnet ein hexadezimale Darstellung der Zahl mit mindestens *width* Zeichen

Beispiel

```
HEXSTR(15, 3) --returns '00F'
```

LEFTSTR

Syntax

```
LEFTSTR(string, count)
```

Beschreibung

Liefert die ersten *count* Zeichen der übergebenen Zeichenkette *string*.

LEN

Syntax

```
LEN(string)
```

Beschreibung

Wie *CHAR_LENGTH*. *CHAR_LENGTH* als Standard SQL ist hier vorzuziehen.

LIKE

Syntax

```
string1 [NOT] LIKE string2
```

Beschreibung

Vergleicht die beiden Zeichenketten wie in Standard SQL definiert. Als Joker Zeichen kann % und _ verwendet werden.

Beispiele

```
'Woolfe' LIKE 'Woo%'
Name LIKE '_oolfe'
Name LIKE '100^% pure orange juice' ESCAPE '^'
```

LOWER

Syntax

```
LOWER(string)
```

Beschreibung

Liefert den String in Kleinbuchstaben.

RIGHTSTR

Syntax

```
RIGHTSTR(string, count)
```

Beschreibung

Liefert die letzten *count* Zeichen der gegebenen Zeichenkette *string*.

STR

Syntax

```
STR(number, width, scale, thousand_separator, fill_character,
decimal_separator)
STR(enumeration_column_reference)
```

Beschreibung

Die erste Variante liefert eine Darstellung der Zahl als Zeichenkette mit den angegebenen Formatierung.

Die zweite Variante liefert eine Darstellung des Aufzählungs-Werts als Zeichenkette.

Beispiel

```
STR(3.14159, 10, 4, ',', '*', '.') --returns ****3.1416
```

SUBSTRING

Syntax

```
SUBSTRING(string FROM start [FOR length])
```

Beschreibung

Liefert den Teilstring der Länge *length* aus *string* angefangen mit dem Zeichen an Position *start*

TRIM

Syntax

```
TRIM([kind [char] FROM] string)
```

Beschreibung

Liefert einen String ohne führende oder abschließende Zeichen.
kind ist eines dieser Schlüsselwörter: LEADING, TRAILING, BOTH
char ist das Zeichen, das entfernt wird. Wird nichts angegeben, werden Leerzeichen entfernt.

Beispiele

die folgenden Ausdrücke liefern alle 'Carl':

```
TRIM('  Carl      ')
TRIM(LEADING FROM '   Carl')
TRIM(TRAILING FROM 'Carl    ')
TRIM(BOTH 'x' FROM 'xxCarlxx')
```

UPPER

Syntax

```
UPPER(string)
```

Beschreibung

Liefert den String in Großbuchstaben.

Siehe auch

Allgemeine Funktionen und Operatoren
Arithmetische Funktionen und Operatoren
String Funktionen und Operatoren
Datum und Zeit Funktionen und Operatoren
Aggregat Funktionen
Sonstige Funktionen und Operatoren

1.3.4.4.13 Datum und Zeit Funktionen und Operatoren

Dies ist eine Liste von Datum und Zeit Funktionen und Operatoren, die in TurboSQL benützt werden können.

+

Syntax

```
date + days
timestamp + days
time + minutes
```

Beschreibung

Addiert eine Anzahl an Tagen zu einem Datum oder einem Zeitstempel. Addiert eine Anzahl an Minuten zu einem Zeitwert.

Beispiele

```
CURRENT_DATE + 1 --Das morgige Datum
CURRENT_TIMESTAMP + 1 --Das morgige Datum mit der aktuellen Zeit
CURRENT_TME + 60 --In einer Stunde
CURRENT_TIME + 0.25 --15 Sekunden später
```

-

Syntax

```
date - days
date1 - date2
timestamp - days
timestamp1 - timestamp2
time - minutes
time1 - time2
```

Beschreibung

Subtrahiert eine Anzahl an Tagen von einem Datum oder Zeitstempel. Subtrahiert eine Anzahl an Minuten von einem Zeit-Wert. Berechnet die Anzahl an Tagen zwischen zwei Datum- oder Zeitstempel-Werten. Berechnet die Anzahl an Minuten zwischen zwei Zeiten.

Beispiele

```
CURRENT_DATE - 1 --Gestern
CURRENT_TIMESTAMP - 1 --Vor 24 Stunden
CURRENT_DATE - DATE'1/1/2006' --Anzahl der Tage seit Anfang 2006
CURRENT_TIME - 60 --Vor einer Stunde
CURRENT_TIME - TIME'12:00 pm' --Anzahl der Stunden seit Mittag (das kann
auch negativ sein)
```

CURRENT_DATE

Syntax

```
CURRENT_DATE
```

Beschreibung

Liefert das aktuelle Datum.

CURRENT_TIME

Syntax

```
CURRENT_TIME
```

Beschreibung

Liefert die aktuelle Zeit auf die Millisekunde genau.

CURRENT_TIMESTAMP

Syntax

```
CURRENT_TIMESTAMP
```

Beschreibung

Liefert den aktuellen Zeitstempel mit einer Genauigkeit von einer Millisekunde. (d.h. *CURRENT_DATE* und *CURRENT_TIME* zusammen)

DATETIMESTR

Syntax

```
DATETIMESTR(TimeStamp, Precision)
```

Beschreibung

Liefert den gegebenen Zeitstempel-Wert als Zeichenkette im Format der aktuellen Ländereinstellung. Die Genauigkeit *Precision* ist 2 für Minuten, 3 für Sekunden und 4 für Millisekunden.

EXTRACT

Syntax

```
EXTRACT(kind FROM date)
```

Beschreibung

Berechnet einen Wert aus *date*. *kind* ist eines der folgenden Schlüsselwörter:

YEAR	Liefert das Jahr.
MONTH	Liefert den Monat.
DAY	Liefert den Tag.
WEEKDAY	Liefert den Wochentag, mit Montag als 1, Dienstag 2 usw.
WEEKDAYNAME	Liefert den Namen des Wochentages in der aktuellen Spracheinstellung.
WEEK	Liefert die Wochennummer wie im ISO standard festgelegt.
HOUR	Liefert die Stunde.
MINUTE	Liefert die Minute.
SECOND	Liefert die Sekunde.
MILLISECOND	Liefert die Millisekunde.

Beispiele

```
EXTRACT(DAY FROM CURRENT_DATE)
EXTRACT(HOUR FROM CURRENT_TIME)
EXTRACT(SECOND FROM CURRENT_TIMESTAMP)
EXTRACT(WEEKDAYNAME FROM CURRENT_DATE)
EXTRACT(MILLISECOND FROM CURRENT_TIME)
EXTRACT(WEEK FROM CURRENT_TIMESTAMP)
```

MAKEDATE

Syntax

```
MAKEDATE(year, month, day)
```

Beschreibung

Liefert den Datum-Wert für das gegebene Datum.

Beispiele

```
SELECT * FROM MyTable WHERE Abs(Today - MakeDate(EXTRACT(YEAR FROM
CURRENT_DATE), EXTRACT(MONTH FROM Birthday), EXTRACT(DAY FROM
```

```
Birthday))) < 7
```

MAKETIMESTAMP

Syntax

```
MAKETIMESTAMP(year, month, day, hour, minute, second, millisecond)
```

Beschreibung

Liefert den Zeitstempel-Wert für das gegebene Datum und die gegebene Zeit.

MAKETIME

Syntax

```
MAKETIME(hour, minute, second, millisecond)
```

Beschreibung

Liefert den Zeit-Wert für die angegebene Zeit.

TIMESTR

Syntax

```
TIMESTR(time, precision)
```

Beschreibung

Liefert die gegebene Zeit als Zeichenkette gemäß der aktuellen Ländereinstellung. Die Genauigkeit *precision* ist 2 für Minuten, 3 für Sekunden und 4 für Millisekunden.

Siehe auch

Allgemeine Funktionen und Operatoren
Arithmetische Funktionen und Operatoren
String Funktionen und Operatoren
Datum und Zeit Funktionen und Operatoren
Aggregat Funktionen
Sonstige Funktionen und Operatoren

1.3.4.4.14 Aggregat Funktionen

Dies ist eine Liste von Aggregat Funktionen, die in TurboSQL verwendet werden können.

AVG

Syntax

```
AVG(column_reference)
```

Beschreibung

Berechnet den Durchschnitt der Werte in der Spalte

COUNT

Syntax

```
COUNT(*|column_reference)
```

Beschreibung

Berechnet die Anzahl der Zeilen in der Spalte

Beispiele

```
COUNT(*)
COUNT(NAME)
```

MAX

Syntax

```
MAX(column_reference)
```

Beschreibung

Berechnet das Maximum der Werte in der Spalte

MIN

Syntax

```
MIN(column_reference)
```

Beschreibung

Berechnet das Maximum der Werte in der Spalte

STDDEV

Syntax

```
STDDEV(column_reference)
```

Beschreibung

Berechnet die Standardabweichung der Werte in den Spalten. Das Argument muss numerisch sein. Das Ergebnis ist immer ein FLOAT.

Beispiel

```
SELECT AVG(Value), STDDEV(Value) FROM Values
```

Kopatibilität

Nur in TurboDB Managed verfügbar.

SUM

Syntax

```
SUM(column_reference)
```

Beschreibung

Berechnet die Summe der Werte in der Spalte

Siehe auch

Allgemeine Funktionen und Operatoren
Arithmetische Funktionen und Operatoren
String Funktionen und Operatoren
Datum und Zeit Funktionen und Operatoren
Aggregat Funktionen
Sonstige Funktionen und Operatoren
Benutzerdefinierte Aggregationen

1.3.4.4.15 Sonstige Funktionen und Operatoren

Dies ist eine Liste diverser Funktionen und Operatoren zum Einsatz in *TurboSQL*.

CONTAINS

Syntax

```
CONTAINS(full-text-search-expression IN table-name.*)
CONTAINS(full-text-search-expression IN column-name1, column-name2,
column-name3, ...)
```

Beschreibung

Liefert für jeden Datensatz true, der dem Volltext-Suchausdruck genügt. In der zweiten Variante muss die Volltext-Suchbedingung der gegebenen Spalten-Untermenge genügen.

Beispiel

Gesucht werden sollen Datensätze in denen das Wort *'computer'*, sowohl in der Spalte *Category* als auch in der Spalte *Name* vorkommt. Das geht so:

```
SELECT * FROM Devices WHERE CONTAINS('computer' IN Category) AND
CONTAINS('computer' IN Name)
```

Kompatibilität

Die zweite Variante gibt es nur in TurboDB Win32.

NEWGUID

Syntax

```
NEWGUID
```

Beschreibung

Erzeugt einen neuen Globally Unique Identifier, wie zum Beispiel {2A189230-2041-44A6-87B6-0AFEE240F09E}.

Beispiel

```
INSERT INTO TABLEA ("Guid") VALUES(NEWGUID)
```

CURRENTRECORDID

Syntax

```
CURRENTRECORDID(table_name)
```

Beschreibung

Liefert den zuletzt verwendeten AutoInc-Wert des gegebenen Tabelle. Mit dieser Funktion ist es möglich, durch mehrere Kommandos innerhalb eines Statements verknüpfte Datensätze in mehrere Tabellen einzutragen.

NULLIF

Syntax

```
NULLIF(value1, value2)
```

Beschreibung

Liefert NULL falls *value1* und *value2* gleich sind und *value1* falls nicht.

Kompatibilität

Nur in TurboDB Managed.

Siehe auch

Allgemeine Funktionen und Operatoren
Arithmetische Funktionen und Operatoren
String Funktionen und Operatoren
Datum und Zeit Funktionen und Operatoren
Aggregat Funktionen
Sonstige Funktionen und Operatoren

1.3.4.4.16 Tabellen Operatoren

TurboSQL unterstützt die folgenden Operatoren um Tabellen zu kombinieren. Sie genügen alle der Standard SQL Spezifikation:

JOIN

Syntax

```
table_reference [INNER | LEFT OUTER | RIGHT OUTER | OUTER] JOIN
table_reference
```

Beispiel

```
SELECT * FROM A JOIN B ON A.a = B.a
```

```
SELECT * FROM A LEFT OUTER JOIN B ON A.a = B.a
```

Beschreibung

Liefert alle Zeilenpaare der beiden Tabellenreferenzen, für die die Bedingung gilt.

UNION

Syntax

```
table_term UNION [ALL] table_term [CORRESPONDING BY column_list]
```

Beispiel

```
TABLE A UNION TABLE B
```

Beschreibung

Liefert alle Zeilen aus den beiden Tabelle. Das Ergebnis ist eindeutig, falls *ALL* nicht definiert ist. Die beiden Tabellen müssen über kompatible Spalten verfügen.

EXCEPT

Syntax

```
table_term EXCEPT [ALL] table_term CORRESPPONDING [BY column_list]
```

Beispiel

```
SELECT * FROM TABLE A EXCEPT SELECT * FROM TABLE B
```

Beschreibung

Liefert alle Zeilen der ersten Tabelle, die nicht in der zweiten Tabelle enthalten sind. Die Ergebnismenge ist eindeutig, falls *ALL* nicht definiert ist. Die beiden Tabellen müssen über kompatible Spalten verfügen.

INTERSECT

Syntax

```
table_primitive INTERSECT table_primitive CORRESPONDING [BY column_list]
```

Beispiel

```
SELECT * FROM TABLE A INTERSECT [ALL] SELECT * FROM TABLE B
```

Beschreibung

Liefert alle Zeilen, die sowohl in der ersten als auch in der zweiten Tabelle enthalten sind. Das Ergebnis ist eindeutig, falls *ALL* nicht definiert ist. Die beiden Tabellen müssen über kompatible Spalten verfügen.

1.3.4.4.17 Unterabfragen

Suchbedingungen in SELECT, INSERT und UPDATE Statements können eingebettete Abfragen beinhalten, die mit einem der folgenden Operatoren mit der Hauptabfrage verglichen werden können. Außerdem kann eine geklammerte Unterabfrage überall da verwendet werden, wo ein Ausdruck erwartet wird. TurboSQL erlaubt sowohl korrelierte als auch unkorrelierte Unterabfragen.

IN

Prüfung ob der Wert eines Ausdrucks in der Ergebnismenge der Unterabfrage gefunden werden kann.

Beispiel

```
select * from SALESINFO
where customerName in (
  select name from CUSTOMER where state = 'CA'
)
```

Liefert alle Verkäufe an Kunden aus Kalifornien und ist im Allgemeinen dasselbe wie

```
select * from SALESINFO join CUSTOMER on customerName = name
where state = 'CA'
```

Exists

Prüft ob die Unterabfrage mindesten einen Datensatz enthält

Beispiel

```
select * from SALESINFO
where exists (
  select * from CUSTOMER
  where name = SALESINFO.customerName and state = 'CA'
)
```

Liefert dasselbe Ergebnis wie das erste Beispiel. Zu beachten ist aber, dass die Unterabfrage jetzt eine Spaltenreferenz auf die äußere Abfrage enthält. Dies wird als korrelierte Unterabfrage bezeichnet.

Any/Some

Überprüft, ob es mindestens einen Datensatz in der Ergebnismenge der Unterabfrage gibt, der die Suchbedingung erfüllt.

Beispiel

```
select * from SALESINFO
where amount > any (
  select averageAmount from CUSTOMER
  where name = SALESINFO.customerName
)
```

Liefert alle Verkäufe, die für den jeweiligen Kunden größer als der Durchschnitt sind.

All

Überprüft, ob alle Datensätze der Ergebnismenge der Unterabfrage der Suchbedingung genügen.

Beispiel

```
select * from SALESINFO
where amount > all (
  select averageAmount from CUSTOMER
  where state = 'CA'
)
```

Liefert die Verkäufe, die für jeden einzelnen Kunden in Kalifornien größer als der Durchschnitt sind.

Unterabfrage als Ausdruck

Eine Unterabfrage in Klammern kann also skalarer Ausdruck verwendet werden. Der Typ des Ausdrucks ist der Typ der ersten Spalte der Ergebnismenge. Der Wert des skalaren Ausdrucks ist Wert der ersten Spalte in der ersten Zeile. Wenn die Ergebnismenge keine Spalte enthält, ist der Ausdruck ungültig. Enthält sie keine Zeile, ist der Wert NULL.

Beispiele

```
select * from [TableB] where C1 like (select C2 from TableB) || '%'
```

```
set A = (select Count(*) from TableA)
```

Kompatibilität

Die Verwendung von Unterabfragen als Ausdrücke ist nur in TurboDB Managed verfügbar.

Siehe auch

WHERE

1.3.4.4.18 Volltextsuche

Volltextsuche ist die Suche nach einem beliebigen Wort in einem Datensatz. Diese Art der Suche ist für Memo- und WideMemo-Felder besonders nützlich, in denen das Suchen mit herkömmlichen Operatoren und Funktionen nicht das erwartete Resultat liefert oder zu lange dauert.

Die Volltextsuche unterliegt zwei Einschränkungen:

- Es muss einen Volltext-Index für die Tabelle geben.
- Eine Volltextsuche bezieht sich immer auf genau eine Tabelle (Es können aber mehrere Volltext-Suchbedingungen in einer WHERE-Klausel sein.).

Die Basis eines Volltext-Index ist das Wörterbuch, das eine normale Datenbank-Tabelle mit einem bestimmten Schema ist. Es enthält die Informationen über indizierte Wörter, ausgeschlossene Wörter, Wort-Relevanz, u.s.w. Sobald das Wörterbuch besteht, kann es für eine beliebige Zahl von Volltext-Indexen auf einer oder auf mehreren Tabellen benutzt werden.

Ab TurboDB 5, werden Volltext-Suchbedingungen in die WHERE Klausel der Abfrage eingebettet:

```
select * from SOFTWARE join VENDOR on SOFTWARE.VendorId = VENDOR.Id
where VENDOR.Country = 'USA' and (contains('office' in SOFTWARE.*) or
contains('Minneapolis' in VENDOR.*))
```

Eine einfache Volltext-Suchbedingung sieht so aus:

```
contains('office -microsoft' in SOFTWARE.*)
```

die zutreffend ist, wenn irgendein Feld des Standard-Volltext-Index der Tabelle SOFTWARE das Wort *office* aber nicht das Wort *microsoft* enthält. Wenn sich die Abfrage auf nur eine Tabelle bezieht, kann dieses auch so geschrieben werden:

```
contains('office -microsoft' in *)
```

Wenn der Volltext-Suchausdruck mehr als ein Wort ohne den Bindestrich enthält, sucht TurboDB nach Datensätzen, die alle gegebenen Wörter enthalten.

```
contains('office microsoft' in SOFTWARE.*)
```

wird daher Datensätze finden, die beide Wörter, *office* und *microsoft,* in einem Feld des Standard-Volltext-Index der Tabelle enthalten.

Nach Wörter, die durch ein Pluszeichen getrennt sind, wird alternativ gesucht. Das Prädikat

```
contains('office star + open' in SOFTWARE.*)
```

findet Datensätze, die das Wort *office* und entweder *star* oder *open* beinhalten (oder beide).

Die Auswertung einer Volltext-Suchbedingung kann auch auf eine Untermenge der indizierten Spalten beschränkt werden:

```
contains('office' in SOFTWARE.Name, SOFTWARE.Category)
```

findet alle Datensätze die das Wort *office* entweder in der Spalte *Name* oder in der Spalte *Category* enthalten. Sollte der Volltex-Index außerdem noch die Spalte *Description* indizieren, wird ein Datensatz nicht geliefert der *office* nur in *Description* enthält. Generell wird die Suchbedingung gegen ndie Gesamtheit der angegebenen Spalten ausgewertet. Enthält die Bedingung einen Ausschlussterm wie in:

```
contains('office -microsoft' in Name, Category)
```

darf das Wort *microsoft* weder in *Name* noch in *Category* enthalten sein, falls es aber in *Description* auftaucht, kann der Datensatz trotzdem Teil der Ergebnismenge sein.

Hinweise

Volltext-Indexe können mit der TurboSQL Anweisung CREATE FULLTEXTINDEX , mit einem der Datenbank-Management Werkzeuge (wie TurboDB5Viewer) oder mit den entsprechenden Methoden der jeweiligen TurboDB Komponenten erstellt werden.

Von TurboDB Tabellen-Level 3 auf Level 4 hat sich die Technologie der Volltext-Indizierung geändert. Die neue Implementierung ist sehr viel schneller und erlaubt sowohl gewartete Indexe als auch Relevanzen. Es wird dringend empfohlen Tabellen-Level 4 zu verwenden, wenn mit

Volltext-Indexen gearbeitet werden soll. Die alte Volltextsuche wird zukünftig eventuell entfernt.

1.3.4.5 Data Definition Language

Turbo SQL bietet die folgenden Kommandos und Datentypen als Teil der DDL:

Kommandos

CREATE TABLE

ALTER TABLE

CREATE INDEX

CREATE FULLTEXTINDEX

UPDATE INDEX/FULLTEXTINDEX

DROP

Datentypen

TurboSQL Datentypen

1.3.4.5.1 CREATE TABLE Befehl

Erzeugt eine neue Tabelle für die aktuelle Datenbank.

Syntax

```
CREATE TABLE table_reference
[LEVEL level_number]
[ENCRYPTION encryption_algorithm]
[PASSWORD password]
[COLLATE collation_name]
(column_definition | constraint_definition [, column_definition |
constraint_definition] ...)
```

wobei eine eine Spaltendefinition *column_definition* so aussieht,

```
column_reference data_type [NOT NULL] [DEFAULT expression | SET
expression]
```

(siehe Datentypen für Tabellenspalten für weitere Informationen)

und eine Gültigkeitsbedingung *constraint_definition* so:

```
PRIMARY KEY (column_reference [, column_reference]...) |
UNIQUE (column_reference [, column_reference]...) |
[CONSTRAINT constraint_name] CHECK (search_condition) |
FOREIGN KEY (column_reference [, column_reference]...)
REFERENCES table_reference (column_reference [, column_reference]...)
[ON UPDATE NO ACTION | CASCADE]
[ON DELETE NO ACTION | CASCADE]
```

Beschreibung

Verwenden Sie CREATE TABLE wenn Sie eine neue Tabelle zur aktuellen Datenbank hinzufügen möchten.

```
CREATE TABLE MyTable (
  OrderNo AUTOINC,
  OrderId CHAR(20) NOT NULL,
  Customer LINK(Customer) NOT NULL,
  OrderDate DATE NOT NULL DEFAULT Today,
  Destination ENUM("Home", "Office", "PostBox"),
  PRIMARY KEY (OrderNo),
  CHECK (LEN(OrderId) > 3),
  FOREIGN KEY (Customer) REFERENCES Customer (CustNo) ON DELETE CASCADE
ON UPDATE NO ACTION
```

```
)
```

Um ein Passwort, einen Schlüssel und eine Sprache zu definieren, fügen Sie die entsprechenden Schlüsselwörter hinzu:

```
CREATE TABLE MyTable
  ENCRYPTION Blowfish PASSWORD 'u(i,iUklah'
  LANGUAGE 'ENU' (Name CHAR(20)
)
```

Die unterstützten Verschlüsselungs-Algorithmen sind in "Data Security" beschrieben.
Der Level kann angegeben werden um rückwärts kompatibel zu bleiben. Falls kein Level angegeben wird, wird das aktuellste Format erzeugt.

Die anderen Klauseln haben ihre Standard-SQL-Bedeutung. Zu beachten ist, daß TurboSQL noch nicht die set null und set default Aktionen für Fremdschlüssel unterstützt, die im SQL-Standard vorhanden ist.

DEFAULT definiert den Vorgabewert der Spalte, der bei Neuanlage eines Datensatzes zugewiesen wird. Verfügbar ab Table Level 4 und höher.

SET definiert einen Ausdruck der jedesmal neu berechnet wird, wenn der Datensatz aktualisiert wird. SET Spalten sind read-only.

Siehe auch

Spalten Datentypen

1.3.4.5.2 ALTER TABLE Befehl

Fügt einer bestehenden Tabelle Spalten hinzu und löscht Spalten aus ihr.

Syntax

```
ALTER TABLE table_reference
[LEVEL level_number]
[ENCRYPTION encryption_algorithm]
[PASSWORD password]
[COLLATE collation_name]
DROP column_reference |
DROP CONSTRAINT constraint_name |
ADD column_definition |
ADD CONSTRAINT constraint_definition |
RENAME column_reference TO column_reference |
MODIFY column_definition
```

Beschreibung

Verwenden Sie die Anweisung ALTER TABLE, um das Schema einer bestehenden Tabelle zu ändern. Die Beschreibungen für column_definition and constraint_definition lesen Sie bitte in CREATE TABLE Befehl. Es gibt sechs verschiedene Optionen:

Löschen einer existierenden Spalte mit DROP:

```
ALTER TABLE Orders DROP Destination
```

column_reference muss sich auf eine existierende Spalte beziehen.

Löschen einer existierenden Gültigkeitsbedingung:

```
ALTER TABLE Orders DROP CONSTRAINT sys_Primary
```

Hinzufügen einer neuen Spalte mit ADD:

```
ALTER TABLE Orders ADD Date_of_delivery DATE
```

Der Name der neuen Spalte darf in der Tabelle noch nicht vorkommen.

Hinzufügen einer neuen Gültigkeitsbedingung mit ADD:

```
ALTER TABLE Orders ADD CONSTRAINT RecentDateConstraint CHECK
(Date_of_delivery > 1.1.2000)
```

```
ALTER TABLE Orders ADD FOREIGN KEY (Customer) REFERENCES Customer
(CustNo)
```

Ändern des Namens einer existierenden Spalte mit RENAME:

```
ALTER TABLE Orders RENAME Date_of_delivery TO DateOfDelivery
```

Die erste *column_reference* ist der Name der existierenden Spalte, die zweite ist der neue Name für diese Spalte. Umbenennen einer Spalte verändert die enthaltenen Daten nicht.

Ändern des Datentyps einer existierenden Spalte mit MODIFY:

```
ALTER TABLE Orders MODIFY DateOfDelivery TIMESTAMP
```

column_reference muss sich auf eine existierende Spalte beziehen. Der Typ der Spalte kann in einen der verfügbaren Datentypen konvertiert werden. Die enthaltenen Daten werden erhalten, falls möglich.

Die Parameter *level_number*, *password*, *key* und *language* haben dieselbe Bedeutung wie in *CREATE TABLE*. Falls *password* und *key* weggelassen werden, bleiben aktuellen Einstellungen erhalten. Um die Verschlüsselung aufzuheben, ist sie auf *NONE* zu setzen.

Diese Anweisung hebt die Verschlüsselung auf:

```
ALTER TABLE Orders ENCRYPTION None
```

Anmerkung: Falls entweder Passwort oder Verschlüsselungsart geändert werden soll, muss aus Sicherheitsgründen beides, Passwort und Verschlüsselung angegeben werden.

Es ist möglich, mehrere Änderungen in beliebiger Reihenfolge in der selben Anweisung zu kombinieren:

```
ALTER TABLE Orders
  ADD Date_of_delivery DATE,
  DROP Destination,
  ADD DeliveryAddress CHAR(200),
  RENAME Customer TO CustomerRef
```

Anmerkung: RENAME und MODIFY sind proprietäre Erweiterungen zu SQL-92.

Kompatibilität:

Die COLLATE-Klausel wird ab Table Level 6 unterstützt.

Die SET Klausel der Spaltendefinition wird erst ab Tabellen-Level 3 unterstützt.

Siehe auch

Spalten Datentypen

1.3.4.5.3 CREATE INDEX Befehl

Erzeugt einen neuen Index für eine bestehende Tabelle.

Syntax

```
CREATE [UNIQUE] INDEX index_reference ON table_reference
(column_reference [ASC|DESC] [,column_reference [ASC|DESC] ...] )
```

Beschreibung

Indexe werden verwendet um Suchaktionen zu beschleunigen. Verwenden Sie CREATE INDEX um einen neuen Index für eine bestehende Tabelle zu erstellen:

```
CREATE INDEX OrderIdIdx ON Orders (OrderId ASC)
```

Sie können auch mehrstufige hierarchische Indexe erstellen:

```
CREATE INDEX TargetDateIdx ON Orders (DeliveryDate DESC, OrderId)
```

Verwenden Sie UNIQUE um einen Index zu definieren, der einen Fehler erzeugt, falls nicht

eindeutige Spaltenwerte in die Tabelle geschrieben werden sollen. Die Voreinstellung ist nicht eindeutig.

Anmerkung

Die Attribute ASC und DESC sind properietäre Erweiterungen des SQL-92 Standards.

1.3.4.5.4 CREATE FULLTEXTINDEX Statement

Erstelle einen neuen Volltext-Index für eine bestehende Tabelle.

Syntax

```
CREATE FULLTEXTINDEX index_reference ON table_reference
(column_reference [, column_reference ...])
DICTIONARY table_reference [CREATE] [UPDATE]
[SEPARATORS '<separating chars>']
```

Beschreibung

Ein Volltext-Index ermöglicht eine Suche mit Volltext-Suchbedingungen wie dem CONTAINS Prädikat. Die Spaltenreferenzen sind eine Liste der Tabellenspalten aller Datentypen mit Ausnahme von Blobs, einschließlich Memos und WideMemos. Volltext-Index benötigen eine zusätzliche Datenbank-Tabelle, die eine Liste der indizierten Wörter enthält, das Wörterbuch.

Die Wörterbuch-Tabelle kann mit diesem Statement oder explizit erstellt werden. Falls CREATE nicht verwendet wird, erwartet das Statement eine Wörterbuch-Tabelle, mit den folgenden Eigenschaften:

- Die erste Spalte ist vom Typ *VARCHAR* oder *VARWCHAR* beliebiger Länge. Diese Spalte nimmt die möglichen Suchbegriffe auf. Falls ein Wort länger ist als es diese Spalte erlaubt, wird es abgeschnitten.
- Die zweite Spalte ist vom Typ *BYTE*. Es nimmt die globale Relevanz des Wortes auf.
- Weitere Spalten können entsprechend den Notwendigkeiten der Anwendung folgen.
- Es muss eine Spalte vom Typ *AUTOINC* geben, um die Wörter zu identifizieren. Die Anzeigeinformation dieser *AUTOINC* Spalte muss die erste Spalte der Tabelle sein.

Falls die CREATE Klausel angegeben wird, erstelle das Statement eine neue Wörterbuch-Tabelle mit einer ersten Spalte vom Typ *VARCHAR(20)*.

Falls UPDATE verwendet wird, werden Worte, die in der Wörterbuch-Tabelle nicht gefunden werden ergänzt. Falls UPDATE nicht angegeben ist, werden nur Wörter der Wörterbuch-Tabelle indiziert und können bei Suchen gefunden werden.

Mit der SEPARATORS Klausel kann definiert werden, wie der Index den Text in einzelne Wörter zerlegt. Beispielsweise ist es manchmal wünschenswert Bindestriche als Teil des Wortes zu betrachten und ein anderes mal als Trennzeichen. Die in <separating chars> definierten Zeichen ersetzen die vorgegebenen Separatoren. Daher müssen alle erwünschten Separatoren angegeben werden, auch die selbstverständlichen wie Leerzeichen, Komma usw.

Hinweis

Die Technologie der Volltext-Suche für Tabellen bis Tabellen-Level 3 unterscheidet sich von der ab -Level 4. Erst ab Tabellen-Level 4 wird automatische Wartung und Berechnung der Relevanz unterstützt.

1.3.4.5.5 DROP Command

Eine Tabelle oder einen Index aus der Datenbank entfernen.

Syntax

```
DROP TABLE table_reference
```

```
DROP INDEX table_reference.index_name
```

```
DROP FULLTEXTINDEX table_reference.index_name
```

Beschreibung

Verwenden Sie DROP um ein Datenbank-Objekt und alle seine zugehörigen Dateien komplett aus der Datenbank zu entfernen.

Beispiele

```
DROP INDEX Orders.OrderIdIdx
DROP TABLE Orders
```

Achtung

Während ein versehentlich gelöschter Index einfach wiederhergestellt werden kann, sind die Daten einer Tabelle nach DROP unwiederbringbar verloren.

1.3.4.5.6 UPDATE INDEX/FULLTEXTINDEX Statement

Repariert einen Index oder Volltext-Index.

Syntax

```
UPDATE INDEX table_reference.index_name
```

```
UPDATE FULLTEXTINDEX table_reference.index_name
```

Beschreibung

Falls ein Index nicht mehr aktuell oder kaputt ist (das kann in der dateibasierten Version von TurboDB passieren, wenn die Anwendung stirbt), kann der Index mit diesem Befehl wiederhergestellt werden. Verwenden Sie den Asterisk * für *index_reference* um alle Indexe der Tabelle wiederherzustellen.

Anmerkung

Die UPDATE FULLTEXTINDEX Anweisung ist nur für den neuen, gewarteten Volltext-Index und dem Tabellen-Level 4 verfügbar.

Beispiel

```
UPDATE INDEX MyTable.*; UPDATE FULLTEXTINDEX MyTable.*
```

1.3.4.5.7 Datentypen für Tabellenspalten

Die folgenden Spaltentypen werden von *TurboSQL* angeboten:

AUTOINC

Syntax

```
AUTOINC(indication) [UNIQUE]
```

TurboDB Spaltentyp

AutoInc

Beschreibung

Integer Feld, bezieht einen eindeutigen Wert von der Datenbank-Engine. Anzeige des Feld-Inhaltes gemäß der Index-Beschreibung. (Siehe "Die Automatic Data Link Technologie".)

Falls UNIQUE definiert wird, wird der Wert der angegeben Indexbeschreibung auf Eindeutigkeit geprüft.

Beispiel

```
AUTOINC('LastName, FirstName')
```

BIGINT

Syntax

```
BIGINT [NOT NULL]
```

TurboDB Spaltentyp

BigInt

Beschreibung

Ganzzahl zwischen –2^63 und +2^63–1

Beispiel

```
BIGINT DEFAULT 4000000000
```

BIT

Nicht unterstützt in TurboSQL.

BOOLEAN

Syntax

```
BOOLEAN [NOT NULL]
```

TurboDB Spaltentyp

Boolean

Beschreibung

Mögliche Werte sind TRUE und FALSE

Beispiel

```
BOOLEAN DEFAULT TRUE
```

BYTE

Syntax

```
BYTE [NOT NULL]
```

TurboDB Spaltentyp

Byte

Beschreibung

Ganzzahl zwischen 0 und 255

Beispiel

```
BYTE NOT NULL DEFAULT 18
```

CHAR

Syntax

CHAR(*n*) [NOT NULL] [COLLATE *collation-name*]

TurboDB Spaltentyp

String

Beschreibung

Ansi String bis N Zeichen. 1 <= N <= 255

Beispiel

```
CHAR(40)
```

Kompatibilität

Die COLLATE Klausel wird ab Tabellen-Level 6 unterstützt.

CURRENCY

Nicht unterstützt in *TurboSQL*, verwenden Sie DOUBLE PRECISION oder BIGINT.

DATE

Syntax

```
DATE [NOT NULL]
```

TurboDB Spaltentyp

Date

Beschreibung

Datum zwischen 1.1.1 und 31.12.9999

Beispiel

```
DATE
```

DECIMAL

Nicht unterstützt in *TurboSQL*, verwenden Sie DOUBLE PRECISION.

DOUBLE PRECISION

Syntax

```
DOUBLE [PRECISION][(p)] [NOT NULL]
```

TurboDB Spaltentyp

Float

Beschreibung

Fließkommazahl zwischen 5.0 x 10^–324 und 1.7 x 10^308 mit 15 signifikanten Stellen. P ist die optionale Anzahl an Nachkommastellen für die Anzeige, die zur Konvertierung von Werten in Zeichenketten benutzt wird.

Beispiel

```
DOUBLE(4) NOT NULL
```

ENUM

Syntax

```
ENUM(value1, value2, value3, ...) [NOT NULL]
```

TurboDB Spaltentyp

Enum

Beschreibung

Ein Wert aus der gegebenen Aufzählung. Die Werte müssen gültige Bezeichner mit maximal 40 Zeichen Länge sein. Die Summe der Werte darf 255 Zeichen nicht überschreiten. Es können maximal 16 Werte angegeben werden.

Beispiel

```
ENUM(Red, Blue, Green, Yellow)
```

FLOAT

Nicht unterstützt in *TurboSQL*, verwenden Sie DOUBLE PRECISION.

GUID

Syntax

```
GUID [NOT NULL]
```

TurboDB Spaltentyp

Guid

Beschreibung

Datentyp für Globally Unique Identifier, 16 Bytes groß

Beispiel

```
GUID DEFAULT '12345678-abcd-abcd-efef-010101010101'
```

INTEGER

Syntax

```
INTEGER [NOT NULL]
```

TurboDB Spaltentyp

Integer

Beschreibung

Ganzzahl zwischen –2.147.483.648 und +2.147.483.647

Beispiel

```
INTEGER NOT NULL
```

LINK

Syntax

```
LINK(table_reference) [NOT NULL]
```

TurboDB Spaltentyp

Link

Beschreibung

Beinhaltet den Wert einer AutoInc-Spalte einer anderen Tabelle und stellt auf diese Weise eine 1:n Beziehung her (Siehe "Die Automatic Data Link Technologie").

Beispiel

```
LINK(PARENTTABLE)
```

LONGVARBINARY

Syntax

```
LONGVARBINARY [NOT NULL]
```

TurboDB Spaltentyp

Blob

Beschreibung

Bit-Stream für beliebige Daten bis 2 GB

Beispiel

```
LONGVARBINARY
```

LONGVARCHAR

Syntax

```
LONGVARCHAR [NOT NULL]
```

TurboDB Spaltentyp

Memo

Beschreibung

Ansi String variabler Länge bis zu 2 G Zeichen

Beispiel

```
LONGVARCHAR
```

Kompatibilität

Die COLLATE Klausel wird ab Tabellen-Level 6 unterstützt.

LONGVARWCHAR

Syntax

```
LONGVARWCHAR [NOT NULL] [COLLATE collation-name]
```

TurboDB Spaltentyp

WideMemo

Beschreibung

Unicode String variabler Länge bis zu 2 G Zeichen

Beispiel

```
LONGVARWCHAR
```

Kompatibilität

Die COLLATE Klausel wird ab Tabllen-Level 6 unterstützt.

MONEY

Nicht unterstützt in *TurboSQL*, verwenden Sie DOUBLE PRECISION oder BIGINT.

NUMERIC

Nicht unterstützt in *TurboSQL*, verwenden Sie DOUBLE PRECISION.

RELATION

Syntax

```
RELATION(table_reference)
```

TurboDB Spaltentyp

Relation

Beschreibung

Beinhaltet die Werte einer beliebigen Anzahl von AutoInc-Spalten einer anderen Tabelle und stellt auf diese Weise eine m:n Beziehung her. (Siehe "Die Automatic Data Link Technologie").

Kompatibilität

Feature wird in TurboDB Managed 1.x nicht unterstützt.

Beispiel

```
RELATION(PARENTTABLE)
```

TIME

Syntax

```
TIME[(p)] [NOT NULL]
```

TurboDB Spaltentyp

Time

Beschreibung

Tageszeit mit einer Genauigkeit von p, wobei p = 2 für Minuten, p = 3 für Sekunden und p = 4 für Millisekunden steht. Die Voreinstellung ist 3. Die Genauigkeit ist erst ab Tabellenlevel 4 verfügbar. Für niedrigere Level gilt p = 2.

Beispiel

```
TIME(4) DEFAULT 8:32:12.002
```

TIMESTAMP

Syntax

```
TIMESTAMP [NOT NULL]
```

TurboDB Spaltentyp

DateTime

Beschreibung

Kombinierter Datentyp für Datum und Zeit mit einer Genauigkeit von Millisekunden zwischen 1.1.1 00:00:00.000 und 31.12.9999 23:59:59.999

Beispiel

```
TIMESTAMP DEFAULT 23.12.1899_15:00:00
TIMESTAMP DEFAULT '5/15/2006 7:00:00'
```

VARCHAR

Syntax

```
VARCHAR(n) [NOT NULL] [COLLATE collation-name]
```

TurboDB Spaltentyp

String

Beschreibung

Wie CHAR

Beispiel

```
VARCHAR(40)
```

Kompatibilität

Die COLLATE Klausel wir ab Tabellen-Level 6 unterstützt.

VARWCHAR

Syntax

```
VARWCHAR(n) [NOT NULL] [COLLATE collation-name]
```

TurboDB Spaltentyp

WideString

Beschreibung

Wie WCHAR

Beispiel

```
VARWCHAR(20) NOT NULL
```

Kompatibilität

Die COLLATE Klausel wir ab Tabellen-Level 6 unterstützt.

SMALLINT

Syntax

```
SMALLINT [NOT NULL]
```

TurboDB Spaltentyp

SmallInt

Beschreibung

Ganzzahl zwischen -32.768 und +32.767

Beispiel

```
SMALLINT
```

WCHAR

Syntax

```
WCHAR(n) [NOT NULL] [COLLATE collation-name]
```

TurboDB Spaltentyp

WideString

Beschreibung

Unicode String bis N Zeichen. Da ein Unicode Zeichen 2 Bytes benötigt, ist die resultierende Feldgröße zwei mal die Anzahl der Zeichen. 1 <= N <= 32767

Beispiel

```
WCHAR(1000) DEFAULT '-'
```

Kompatibilität

Die COLLATE Klausel wir ab Tabellen-Level 6 unterstützt.

1.3.4.6 Programmiersprache

Dieses Feature ist nur in TurboDB Managed verfügbar.

TurboSQL verfügt über Spracheelemente zur Erstellung von Routinen, die in SQL Befehlen verwendet werden können.

Benutzerdefinierte Funktionen

Funktionen können SQL Kommandos vereinfachen oder diese um zusätzliche Funktionalität erweitern. Sie werden entweder in TurboSQL geschrieben oder aus einer .NET Assembly importiert. Funktionen können über Input-Parameter verfügen und berechnen immer einen Rückgabewert. Sie dürfen keine Seiteneffekte haben. Funktionen werden mit den Statements CREATE FUNCTION und DROP FUNCTION verwaltet. Sie können für berechnete Indexe, berechnete Spalten und Gültigkeitsbedingungen verwendet werden.

Benutzerdefinierte Prozeduren

Prozeduren werden verwendet um komplexe Sequenzen von SQL Befehlen in einem einzigen Statement aufzurufen. Zum Beispiel können durch Verwendung einer Prozedur mehrere Datensätze verschiedener Tabellen in einem Schritt geändert werden. Prozeduren werden entweder in TurboSQL geschrieben oder aus einer .NET Assembly importiert. Prozeduren werden mit den Statements CREATE PROCEDURE und DROP PROCEDURE verwaltet.

Benutzerdefinierte Aggregate

Aggregate berechnen kumulierten Werte in Gruppen der Ergebnismenge. Sie können beispielsweise verwendet werden um das zweitgrößte Maximum, die Standardabweichung oder andere akkumulierte Werte aus einer gruppierten Ergebnismenge zu berechnen. Aggregate werden in einer .NET Assembly implementiert und mit den Statements CREATE AGGREGATE und DROP AGGREGATE verwaltet.

Statements

CREATE FUNCTION Statement

CREATE PROCEDURE Statement

CREATE AGGREGATE Statement

DROP FUNCTION/PROCEDURE/AGGREGATE Statement

DECLARE Statement

SET Statement

WHILE Statement

IF Statement

CALL Statement

Weiter Themen

Parameter mit .NET Assemblies austauschen

1.3.4.6.1 CALL Statement

Syntax

```
CALL procedure_name([argument, ...])
```

Beschreibung

Führt eine Stored Procedure mit den gegebenen Argumenten aus.

Beispiel

```
CALL LogLine('Das ist ein Test')
```

Kompatibilität

Diese Anweisung ist nur in TurboDB Managed verfügbar.

1.3.4.6.2 CREATE FUNCTION Statement

Syntax

```
CREATE FUNCTION function_name([:parameter_name data_type]...) RETURNS
data_type AS RETURN expression
```

```
CREATE FUNCTION function_name([:parameter_name data_type]...) RETURNS
data_type AS BEGIN statement [statement]... END
```

```
CREATE FUNCTION function_name([:parameter_name data_type]...) RETURNS
data_type AS EXTERNAL NAME [namespace_name.class_name].method_name,
assembly_name
```

Beschreibung

Eine Funktion kann überall da eingesetzt werden, wo ein Skalar erwartet wird, in Select Elementen, Suchbedingungen und anderen Ausdrücken.

namespace_name.class_name bezeichnet eine öffentliche (public) Klasse der Assembly.

method_name bezeichnet den Namen einer statischen public Funktion der Assembly. Die Funktion muss öffentlich (public) und nicht überladen (overloaded) sein . Die Parameter müssen mit denen der TurboSQL Funktion übereinstimmen (siehe "Parameter mit .NET Assemblies austauschen").

Beispiel

```
CREATE FUNCTION LastChar(:S WCHAR(1024)) RETURNS WCHAR(1) AS
RETURN SUBSTRING(:S FROM CHAR_LENGTH(:s) FOR 1)
```

```
CREATE FUNCTION Replicate(:S WCHAR(1024), :C INTEGER)RETURNS WCHAR(1024)
AS BEGIN
     DECLARE :I INTEGER
     DECLARE :R WCHAR(1024)
     SET :I = 0
     SET :R = ''
     WHILE :I < :C BEGIN
          SET :R = :R + :S
          SET :I = :I + 1
     END
     RETURN :R
END
```

```
CREATE FUNCTION CubicRoot(:X FLOAT) RETURNS FLOAT AS
EXTERNAL NAME [MathRoutines.TurboMath].CubicRoot,MathRoutines
```

Kompatibilität

Diese Anweisung ist nur in TurboDB Managed verfügbar.

1.3.4.6.3 CREATE PROCEDURE Statement

Syntax

```
CREATE PROCEDURE function_name([:parameter_name data_type]...) AS
statement
```

```
CREATE PROCEDURE function_name([:parameter_name data_type]...) AS BEGIN
statement [statement]... END
```

```
CREATE PROCEDURE function_name([:parameter_name data_type]...) AS
EXTERNAL NAME [namespace_name.class_name].method_name,assembly_name
```

Beschreibung

namespace_name.class_name bezeichnet eine öffentliche (public) Klasse der Assembly.

method_name bezeichnet den Namen einer statischen public Funktion der Assembly. Die Funktion muss öffentlich (public) und nicht überladen (overloaded) sein . Die Parameter müssen mit denen der TurboSQL Funktion übereinstimmen (siehe "Parameter mit .NET Assemblies austauschen").

Beispiel

```
CREATE PROCEDURE Insert(:LastName WCHAR(40), :Salary INTEGER, :
Department WCHAR(40) AS BEGIN
     INSERT INTO Departments ([Name]) VALUES(:Department)
     INSERT INTO Employees (LastName, Salary, Department) VALUES(:
LastName, :Salary, :Department)
END
```

Das folgende Beispiel setzt die öffentliche, statische Methode *Send* der öffentlichen Klasse *SmtpClient* im Namensraum (namespace) *Email* der Assembly *Email.dll* voraus.Die Assembly muss im Verzeichnis der Datenbankdatei (tdbd) untergebracht sein.

```
CREATE PROCEDURE SendEmail(:Address WCHAR(100), :Subject WCHAR(100), :
Content WCHAR(100)) AS
EXTERNAL NAME [Email.SmtpClient].Send,Email
```

Kompatibilität

Diese Anweisung ist nur in TurboDB Managed verfügbar.

1.3.4.6.4 CREATE AGGREGATE Statement

Syntax

```
CREATE AGGREAGATE aggregate_name([:parameter_name data_type]...) RETURNS
data_type AS EXTERNAL NAME [namespace_name.class_name],assembly_name
```

Beschreibung

namespace_name.class_name bezeichnet eine öffentliche (public) Klasse der Assembly. Diese Klasse muss über einen Default Konstruktor verfügen und drei Methoden implementieren:

```
[C#]
public <class_name>()
public void Init()
public void Accumulate(<data_type>)
public <data_type> Terminate()
```

data_type muss mit den Paramentern einer TurboSQL Funktion übereinstimmen (siehe " Parameter mit .NET Assemblies austauschen").

Wenn ein benutzerdefiniertes Aggregat in einem SQL Statement verwendet wird, z.B: *SELECT MAX2(Salary) FROM Employees*, erzeugt die Engine eine Instanz der CLR Aggregat Klasse. Zu Beginn jeder Gruppe wird die *Init* Methode aufgerufen. Danach erfolgt für jeden Datensatz der Gruppe der Aufruf von *Accumulate* in der in der GROUP BY Klausel definierten Reihenfolge. Nachdem alle Datensätze der Gruppe akkumuliert sind, führt die Engine die Funktion *Terminate* aus, um das Ergebnis abzurufen. Jede angeforderte Ressource kann in *Terminate* freigegeben

werden.

Beispiel

Der C# Code für ein Aggregat zur Berechnung des Maximums zweiter Ordnung sieht so aus:

```
namespace MathRoutines {

     public class SecondOrderMax {

          public SecondOrderMax() { }

          public void Init() {
               max = null;
               max2 = null;
          }

          public void Accumulate(double? x) {
               if (max == null || x > max) {
                    if (max != null)
                         max2 = max;
                    max = x;
               } else if (max2 == null || x > max2)
                    max2 = x;
          }

          public double? Terminate() {
               return max2;
          }

          private double? max;
          private double? max2;
     }
}
```

Und so wird das Aggregat in TurboSQL importiert:

```
CREATE AGGREGATE Max2(:x DOUBLE) RETURNS DOUBLE AS
EXTERNAL NAME [MathRoutines.SecondOrderMax],MathRoutines
```

Kompatibilität

Diese Anweisung ist nur in TurboDB Managed verfügbar.

1.3.4.6.5 DROP FUNCTION/PROCEDURE/AGGREGATE Statement

Syntax

DROP FUNCTION | PROCEDURE | AGGREGATE

Beschreibung

Verwenden Sie DROP um die Funktion, Prozedur oder das Aggregat aus der Datenbank zu löschen.

Kompatibilität

Diese Anweisung ist nur in TurboDB Managed verfügbar.

1.3.4.6.6 DECLARE Statement

Syntax

```
DECLARE :variable_name data_type
```

Beispiel

```
DECLARE :i INTEGER
DECLARE :s CHAR(300)
```

Kompatibilität

Diese Anweisung ist nur in TurboDB Managed verfügbar.

1.3.4.6.7 IF Statement

Syntax

```
IF search_condition THEN if_statement [ELSE else_statement]
```

Beschreibung

Führt das Statement *if_statement* aus falls und nur falls die Bedingung *search_condition* wahr (True) ist . Falls *search_condition* nicht True liefert und *else_statement* gegeben ist, wird dieses Statement ausgeführt.

Beispiel

```
IF :s <> '' THEN SET :s = :s + ', '
```

```
IF :a > 0 THEN BEGIN
     SET :r = SQRT(:a)
END ELSE BEGIN
     SET :r = SQRT(-:a)
END
```

Kompatibilität

Diese Anweisung ist nur in TurboDB Managed verfügbar.

1.3.4.6.8 SET Statement

Syntax

```
SET :variable_name = expression
```

Beschreibung

Weist der Variable einen neuen Wert zu.

Beispiel

```
SET :LastName = 'Miller'
```

Kompatibilität

Diese Anweisung ist nur in TurboDB Managed verfügbar.

1.3.4.6.9 WHILE Statement

Syntax

```
WHILE search_condition statement
```

Beschreibung

Führt *statement* aus, so lange *search_condition* wahr (True) ist.

Beispiel

```
DECLARE :I INTEGER
WHILE :I < 100 SET :I = :I + 1
```

```
DECLARE :I INTEGER
WHILE :I < 100 BEGIN
     SET :I = :I + 1
END
```

Kompatibilität

Diese Anweisung ist nur in TurboDB Managed verfügbar.

1.3.4.6.10 Parameter mit .NET Assemblies austauschen

Assemblies, die externe Funktionen, Prozeduren oder Aggregate zur Verfügung stellen, müssen im Verzeichnis der Datenbankdatei untergebracht sein.

Beim Aufruf von Methoden aus .NET Assemblies werden die Paramter von TurboSQL nach CLR gemappt, gemäß der nachfolgenden Tabelle. Wie zu sehen ist, besteht ein Unterschied zwischen Funktionen auf der einen und Prozeduren und Aggregaten auf der anderen Seite.

Funktionen haben nur Input-Parameter und es wird NULL zurückgegeben, falls eines der Argumente NULL ist. In diesem Fall wird die CLR Methode überhaupt nicht ausgeführt. Aus diesem Grund wird NULL nie an eine benutzerdefinierte CLR Funktion übergeben und die CLR Methode darf nur mit CLR-Typen deklariert werden, die keine NULL-Werte kennen (non-nullable). Zum Beispiel:

```
CREATE FUNCTION Log2(:x DOUBLE NOT NULL) RETURNS DOUBLE NOT NULL AS
EXTERNAL NAME [MyNamespace.MyClass].MyMethod,MyAssembly
```

korrespondiert mit dieser Definition in C#:

```
public class MyClass {
     static public double MyMethod(double x) {...}
}
```

Weil das Argument nicht zwischen 0 und NULL unterscheidet, kann *Log2* nie mit NULL-Werten verwendet werden. Sieht die Deklaration dagegen so aus

```
CREATE FUNCTION Log2(:x DOUBLE) RETURNS DOUBLE AS EXTERNAL NAME
[MyNamespace.MyClass].MyMethod,MyAssembly
```

ist obige Definition gültig in C# . Wird Log2 mit einem NULL-Argument aufgerufen, liefert die Engine NULL ohne die CLR Methode aufzurufen.

Diese Regel gilt nur für Funktionen; CLR Prozeduren und Aggregate werden immer aufgerufen, auch für Argumente die NULL sind. Daher müssen die Parameter und der Rückgabetyp der CLR Definition in der Lage sein, NULL-Werte zu transportieren. TurboSQL tut das indem *null* (*Nothing* in Visual Basic) übergeben wird, was für *CHAR*- und *LONGVARBINARY*-Datentypen offensichtlich ist. Um auch mit anderen Datentypen wie *Int64* oder *DateTime* in dieser Weise verfahren zu können, muss der Nullable-Wrapper verwendet werden.

```
CREATE PROCEDURE TestProc(:x DOUBLE) AS EXTERNAL NAME [MyNamespace.
MyClass].MyMethod,MyAssembly
```

wird daher abgebildet durch

```
public class MyClass {
     static public void MyMethod(Nullable<double> x) {...}
}
```

oder

```
public class MyClass {
     static public void MyMethod(double? x) {...}
}
```

oder in Visual Basic:

```
Public Class MyClass
     Public Shared Sub MyMethod(X As Nullable(Of Double))
          ...
     End Sub
End Class
```

Weitere Informationen zu Nullable-Wrapper sind in der MSDN zu finden, als Suchbegriff kann *Nullable class* verwendet werden.

TurboSQL Typ	Non-Nullable CLR Typ	Nullable CLR Typ
BYTE	*System.Int64*	*Nullable<System.Int64>*
SMALLINT	*System.Int64*	*Nullable<System.Int64>*
INTEGER	*System.Int64*	*Nullable<System.Int64>*
BIGINT	*System.Int64*	*Nullable<System.Int64>*
FLOAT	*System.Double*	*Nullable<System.Double>*
AUTOINC	*System.Int64*	*Nullable<System.Int64>*
BOOLEAN	*System.Int64 (0 corresponds to false, all other values to true)*	*Nullable<System.Int64>*
ENUM	*System.Int64*	*Nullable<System.Int64>*
GUID	*System.Guid*	*Nullable<System.Guid>*
LINK	*System.Int64*	*Nullable<System.Int64>*
RELATION	*System.Int64[]*	*System.Int64[]*
CHAR(X), VARCHAR (X)	*System.String*	*System.String*
WCHAR(X), VARWCHAR(X)	*System.String*	*System.String*
LONGVARCHAR	*System.String*	*System.String*
LONGVARWCHAR	*System.String*	*System.String*
LONGVARBINARY	*System.Byte[]*	*System.Byte[]*
TIME	*System.DateTime*	*Nullable<System.DateTime>*
DATE	*System.DateTime*	*Nullable<System.DateTime>*
TIMESTAMP	*System.DateTime*	*Nullable<System.DateTime>*

Kompatibilität

Diese Anweisung ist nur in TurboDB Managed verfügbar.

1.3.5 TurboDB Produkte und Werkzeuge

Es gibt verschiedene Werkzeuge, die Ihnen die Arbeit mit TurboDB erleichtern.

TurboDB Viewer: Visuelles Werkzeug zur Verwaltung und Bearbeitung von TurboDB Datenbanken, Tabellen und Indexen.

TurboDB Pilot: SQL Konsole zur Verwaltung und Bearbeitung von Single-File Datenbanken ab Tabellen Level 5.

dataWeb Compound File Explorer: Visuelles Verwaltungswerkzeug für TurboDB Single-File Datenbanken

TurboDB Workbench: Konsolenprogramm zum Bearbeiten von TurboDB Tabellen und Indexen.

TurboDB Data Exchange: Konsolenprogramm zum Importieren und Exportieren von Daten in und aus TurboDB Tabellen.

TurboDB Server: Datenbankserver für TurboDB-Tabellen: Erlaubt bis zu 100 gleichzeitige Sessions auf eine Datenbank (bald verfügbar).

TurboDB Studio: RAD Werkzeug für die Entwicklung von Windows Anwendungen mit TurboDB Engine.

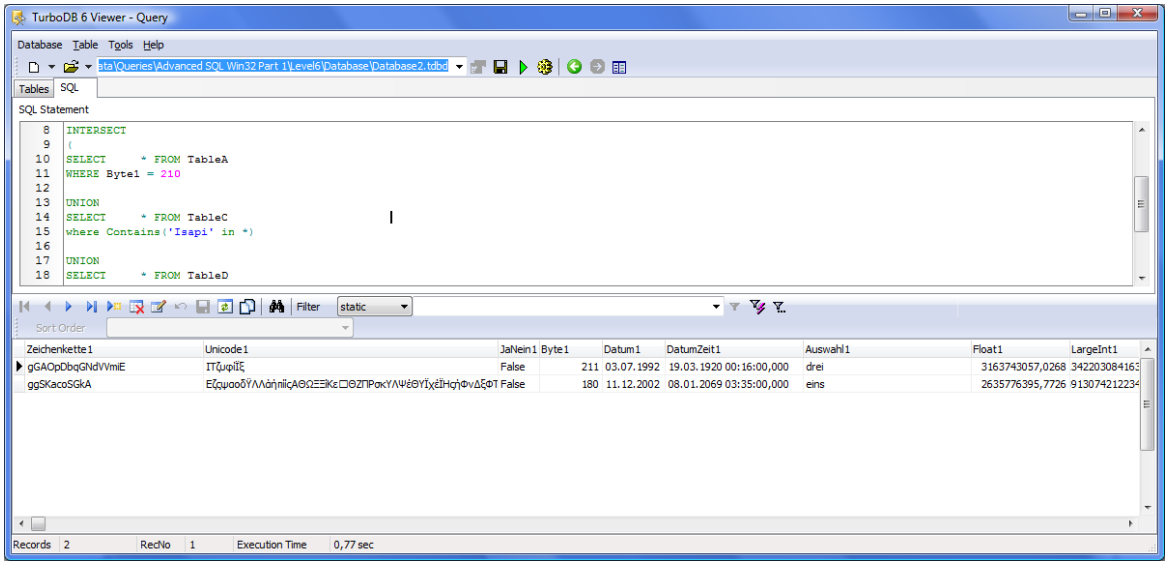
Weitere Information zu TurboDB Produkten und Tools finden Sie auf der dataweb Homepage www.dataweb.de.

1.3.5.1 TurboDB Viewer

TurboDB Viewer ist ein visuelles Werkzeug um Tabellen und Indexe zu verwalten und um Tabellen zu editieren. Das können Sie mit dem Programm machen:

- Tabelleninhalte betrachten und editieren
- SQL Abfragen ausführen
- Datenbanktabellen entwerfen
- Datenbanktabellen ändern
- Indexe erstellen. ändern oder löschen
- Volltexindexe erstellen. ändern oder löschen

TurboDB Viewer ist im Lieferumfang von TurboDB enthalten und unterstützt alle Tabellen Level und Features. Sie finden ihn entweder im Startmenü oder als ausführbare Datei namens *TurboDB6Viewer.exe*.

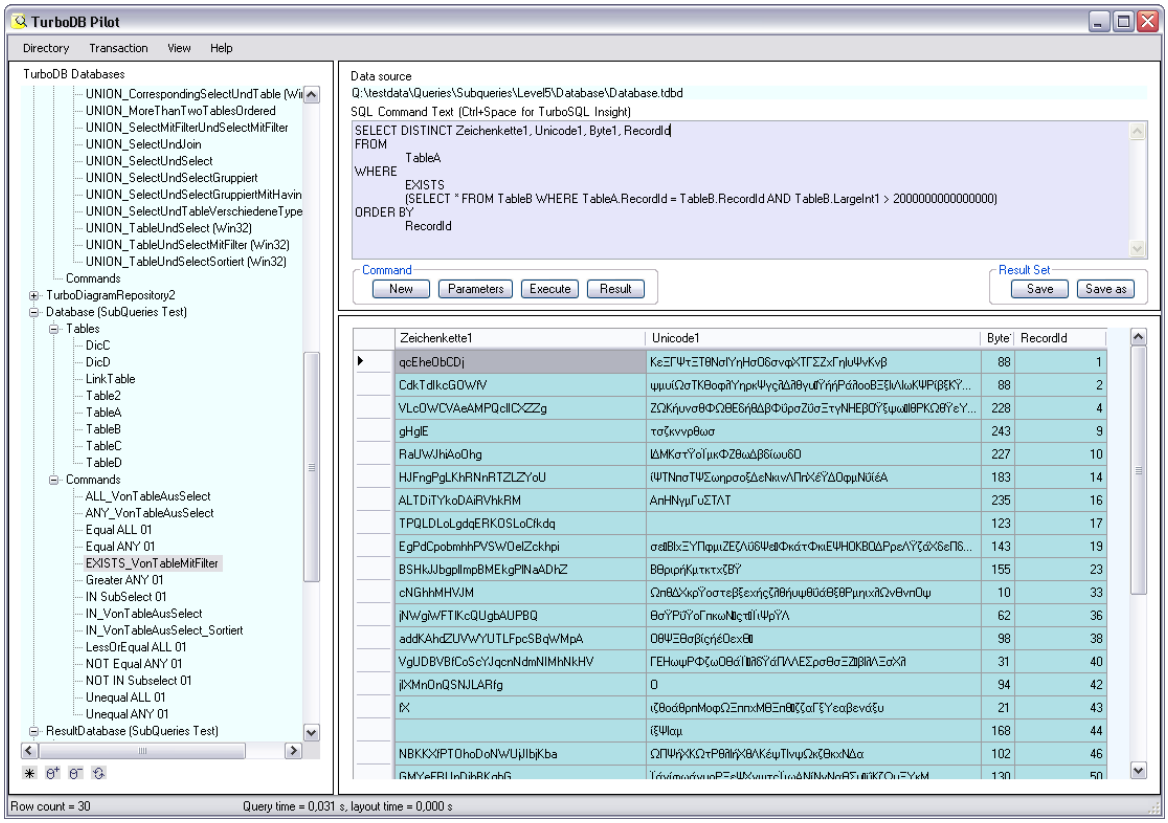


Beim Erstellen von Datenbanken für TurboDB Managed muss beachtet werden, dass nur Features verwendet werden, die TurboDB Managed unterstützt werden. Daher ist für TurboDB Managed der Einsatz von TurboDB Pilot empfehlenswert, der selbst auf der Managed Datenbank Engine basiert.

1.3.5.2 TurboDB Pilot

TurboDB Pilot ist eine SQL Konsole für TurboDB Level 5 Datenbanken, der auf der Managed Database Engine basiert. Es ist ein vollständiges Werkzeug um TurboDB Tabellen Level 5 zu verwalten.

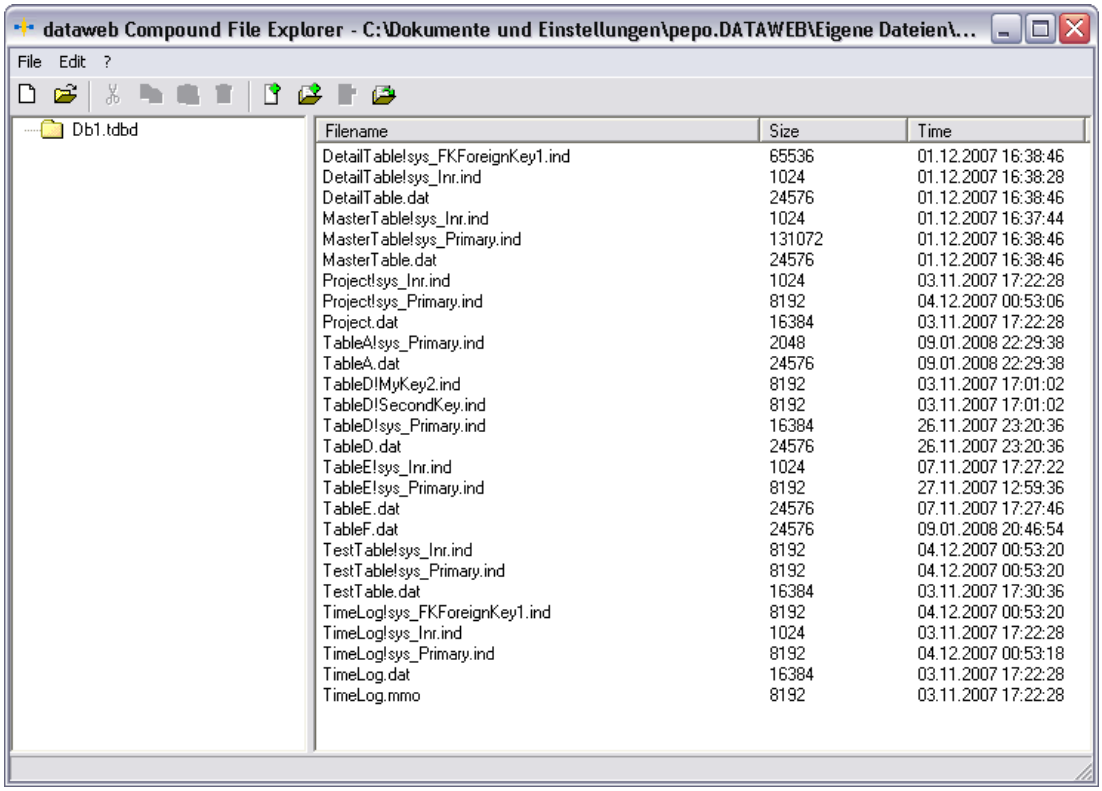
- Erzeugen und Ändern von Tabellen
- Abfragen mit Parametern ausführen
- Update von Datenbanken
- Anzeige von Datenbank Schema Informationen



TurboDB Pilot speichert ein Verzeichnis aller Datenbanken ihres Systems und erleichtert damit das Wiederfinden.

1.3.5.3 dataweb Compound File Explorer

Dies ist das Management Werkzeug zu dataweb Compound File-Technologie. Da eine TurboDB Datenbank-Datei ein Compound File ist, kann mit diesem Werkzeug der Inhalt einer Datenbank-Datei angesehen und editiert werden. Es wird mit allen TurboDB Produkten ausgeliefert.



1.3.5.4 TurboDB Workbench

tdbwkb ist ein kleines textbasiertes Freeware Tool zur Verwaltung von TurboDB Tabellen. Es ist für Windows erhältlich und kann auch von unserer Website http://www.turbodb.de heruntergeladen werden

- Tabellen erzeugen, ändern, umbenennen, löschen
- Tabelleninhalte betrachten
- Tabellenstruktur ansehen und Tabelle reparieren
- Indexe für Tabellen erstellen, ändern und löschen

Beim Start des Programms wird das *tdbwkb* Prompt angezeigt, hier können Sie die verschiedenen Kommandos eingeben. Geben Sie help ein um eine Liste mit den möglichen Befehlen anzuzeigen.

Hier beispielhaft eine *tdbwkb* Session um die verfügbaren Features zu demonstrieren.

```
c:\programme\dataweb\TurboDBStudio\bin\tdbwkb
dataWeb Turbo Database Workbench Version 4.0.1 (TDB 6.1.6)
Copyright (c) 2002-2005 dataWeb GmbH, Aicha, Germany
Homepage http://www.dataWeb.de, Mail dataWeb Team
Type 'help' to get a list of available commands.

tdbwkb> help
  Abbreviations are not allowed. The commands are:
  altertable    Modifies an existing table.
  bye           Ends the tdbwkb session.
  cd            Changes the current directory.
  debug         Toggles debug mode. (Debug mode prints log messages.)
```

```
  delindex      Deletes an index from a table.
  deltable      Deletes all files of a table.
  help          Prints this list of commands.
  newftindex    Create a new full text index for a table.
  newindex      Creates a new index for a table.
  newtable      Creates a new table.
  pwd           Prints current working directory.
  show          Shows a rough preview of the table.
  switchdb      Opens another database.
  rename        Renames a table.
  repair        Rebuilds a table and all its indexes.
  tableinfo     Shows the description of a table.
  Type help <cmd> to get more specific help for a command.
  Note: You may also use tdbwkb in batch mode by appending the command
directly
  to the call. Example:
  tdbwkb tableinfo mytable

tdbwkb> newtable animals
S40Name,A'Land,Water,Air'Area,PImage,MDescription,N'Name'RecordId
Creating table animals.dat with these columns:
  1 S40  Name
  2 A    Area, Values = Land,Water,Air
  3 P    Image
  4 M    Description
  5 N    RecordId
tdbwkb> tableinfo animals
Retrieving structure of table animals.dat...
Table columns:
  1 S40  Name
  2 A    Area, Values = Land,Water,Air
  3 P    Image
  4 M    Description
  5 N    RecordId
Indexes:
animals.inr                   RecordId:4
animals.id                    Name:40

tdbwkb> altertable animals n2=S40Family
Restructuring table animals.dat to these columns:
  1 S40  Name
  2 S40  Family
  3 A    Area, Values = Land,Water,Air
  4 P    Image
  5 M    Description
  6 N    RecordId

tdbwkb> newindex animals byfamily Family,Name

tdbwkb> tableinfo animals
Retrieving structure of table animals.dat...
Table columns:
  1 S40  Name
  2 S40  Family
  3 A    Area, Values = Land,Water,Air
  4 P    Image
  5 M    Description
  6 N    RecordId
Indexes:
animals.inr                   RecordId:4
animals.id                    Name:40
byfamily.ind                  Family:40, Name:40
```

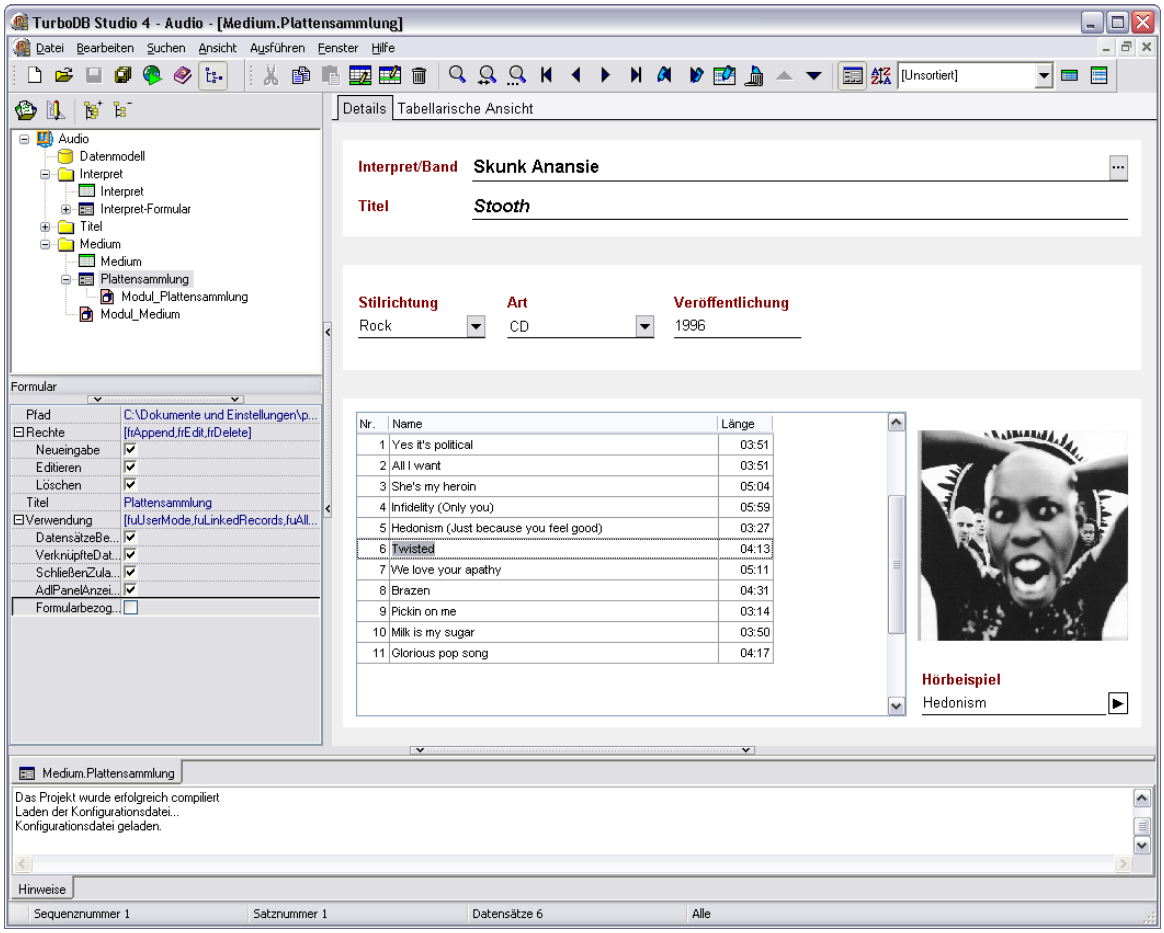
1.3.5.5 TurboDB Studio

Das ist unser Werkzeug um Turbo Datenbank-Anwendungen unter Windows zu erzeugen. Mit TurboDB Studio erzeugen Sie Formulare und Berichte für interaktive Programme und Ausdrucke.

Sie erstellen in wenigen Stunden Ihre eigenen Anwendungen (Exe) mit eigenem Namen, Logo, Menü und Werkzeugleiste. TurboDB Entwickler könnnen mit TurboDB Studio Ihre TurboDB Datenbank auf einfachste Weise verwalten, Daten eingeben und Berichte drucken.

Mit TurboDB Studio können Sie

- TurboDB Datenbanken komfortabler verwalten als mit TurboDB Viewer
- Prototypen für TurboDB Anwendungen sehr schnell erstellen
- Ihren Kunden ein zusätzliches Werkzeug zur Verfügung stellen (einschließlich Berichtwesen)
- Vollständige Windows Anwendungen basierend auf TurboDB erstellen



Der Vorgänger von TurboDB Studio hieß Visual Data Publisher. Weitere Informationen zu TurboDB Studio finden Sie unter http://www.dataweb.de

1.3.5.6 TurboDB Data Exchange

Ein konsolenbasiertes Freeware Tool, das Daten von und in TurboDB Tabellen lesen und schreiben kann. Sie können die aktuelle Version jederzeit von der dataweb Homepage herunterladen.

Von TurboDB Data Exchange unterstützte Formate sind Text, dBase, TurboDB, XML und ADO.

Index

- C -

- D -

- E -

- F -

- G -

- H -

- I -

- J -

- K -

- L -

- M -

- N -

- O -

- P -

- Q -

- R -

- S -

- U -

- U -

- V -

- W -

- Y -

- Z -