TurboDB.NET

The Programmer's Guide to Powerful .NET Database Applications

TurboDB.NET

True .NET feeling for your database applications

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.

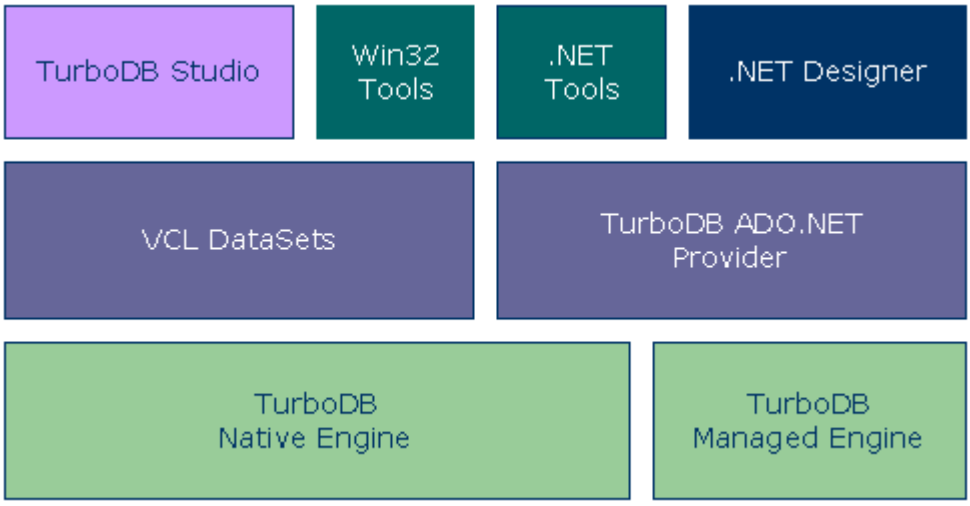
TurboDB

Printed: Oktober 2009 in Aicha, Germany

Table of Contents

1 TurboDB Documentation

TurboDB is a family of powerful and reliable database engines and tools.



At the basis, there are two database kernel implementations. One is for native Win32 and therefore compiled to machine language, the other is for .NET and consists of 100% managed code. When programming on the .NET platform, you can use either one with different advantages:

- The Managed Engine is more secure due to the features of the Common Language Runtime and does not require special execution rights.
- The Managed Engine runs also on .NET Compact Framework, which includes mobile devices like PDAs and smartphones.
- The Managed Engine supports user-defined functions, stored procedures and user-defined aggregates.
- The Native Engine supports multiple concurrent users like Access and Paradox do.
- The Native Engine includes features like sub-queries and full-text indexes, which the Managed Engine does not have.
- The Native Engine supports also former database formats while the Managed Engine just works with single-file databases and the current table level 5.

This manual refers to both engines as far as .NET applications are concerned, i.e. it covers the ADO.NET Provider related side of the above picture. VCL Datasets are used for the Borland development tools and described in another manual.

This document serves as the manual for two TurboDB products, **TurboDB 5 for .NET**, which is based on the native engine and **TurboDB Managed**, which is based on the managed engine. Both of them contain the database engine, an ADO.NET data provider and a set of tools for managing TurboDB databases.

TurboDB for .NET requires the .NET framework and can be used in any .NET development environment with any .NET enabled programming language like C#, VB.NET, Delphi for .NET, Jscript, J# etc. Some visual design features might however not be available in each environment. Note that TurboDB Managed currently only runs on .NET 2.0, while TurboDB for .NET also supports .NET 1.1.

This documentation is divided in four parts. After some instructions installation, upgrading, licensing etc, the respective database engine is described in "Database Engine", this book includes the reference of the TurboDB SQL dialect called TurboSQL and explains many database related aspects. The third part "Developing with TurboDB" describes typical development tasks and how they are solved with your TurboDB product. The forth part, "ADO.NET Data Provider Reference" is the reference for the native ADO.NET provider of TurboDB.

1.1 Configuring the Provider Factory

.NET Framework 2.0 allows the use of generic database factories. If you want to use this programming feature, you must add TurboDB to the *machine.config* under *system.data/DbProviderFactories.* Set the version according to the one you have installed (look it up in the assembly cache, if you are not sure, which version you have). The entry looks like this:

```
<system.data>
     <DbProviderFactories>
          <add name="Odbc Data Provider" invariant="System.Data.Odbc"
description=".Net Framework Data Provider for Odbc"
type="System.Data.Odbc.OdbcFactory, System.Data, Version=2.0.0.0,
Culture=neutral, PublicKeyToken=b77a5c561934e089" />
          <!-- Other provider factories here>

          <!-- TurboDB Win32 for ADO.NET 5>
          <add name="dataWeb TurboDB Data Provider"
invariant="DataWeb.TurboDB" description=".NET Framework Data Provider
for TurboDB" type="DataWeb.TurboDB.TurboDBFactory,
DataWeb.TurboDB.Native20.Provider, Version=X.X.X.X, Culture=neutral,
PublicKeyToken=ee707c084c1d234c" />

          <!-- TurboDB Managed>
          <add name="dataWeb TurboDB Managed Data Provider"
invariant="DataWeb.TurboDBManaged" description=".NET Framework Data
Provider for TurboDB Managed"
type="Dataweb.TurboDBManaged.TurboDBFactory,
Dataweb.TurboDB.Managed20.Provider, Version=X.X.X.X, Culture=neutral,
PublicKeyToken=e71a100181b21fc6" />

     </DbProviderFactories>
</system.data>
```

Note

Do not add factory provider entries for providers not installed on your system.

1.2 Licensing TurboDB Win32 for ADO.NET

Applications using TurboDB Win32 always must be licensed. The license is stored in a *Dataweb.TurboDB.lic* file in the installation folder of TurboDB. If you drop at least one *TurboDBConnection* component on a Windows form or ASP.NET form in Visual Studio, the license is added to your project automatically. If you use TurboDB only from your code or if you insert TurboDB component via DDEX into your application, an exception is thrown when *TurboDBConnection* is created for the first time within the application. The exception tells you that the application has been compiled without TurboDB license information.

To include license information for TurboDB Win32 in your application:

1. Check whether your project already contains a file named *licenses.licx*. Switch on *Show All Files* in the *Solution Explorer* to do this. If the file does not yet exist, add a new text file with this name. (In VS 2005, one usually puts it in the *Properties* folder.)
2. Open this text file and add the following line:

   For TurboDB Win32 5 for ADO.NET 1.1:

   ```
   DataWeb.TurboDB.TurboDBConnection, DataWeb.TurboDB, Version=<main
   version>.<subversion>.0.0, Culture=neutral,
   PublicKeyToken=ee707c084c1d234c
   ```

   For TurboDB Win32 5 for ADO.NET 2.0:

   ```
   DataWeb.TurboDB.TurboDBConnection,
   DataWeb.TurboDB.Native20.Provider, Version=<main
   ```

```
version>.<subversion>.0.0, Culture=neutral,
PublicKeyToken=ee707c084c1d234c
```

Note that you can look up the exact version and public key token from a view of the assembly cache (for example in the Administrative Tools: *.NET Framework X Configuration/Manage the Assembly Cache/View List of Assemblies in the Assembly Cache*). Another source for these entries is the *licenses.txt* file in the bin folder of your TurboDB installation.

Set <main version> and <subversion> to the actual version you have. For example, to 1 and 1 for TurboDB Managed or to 5 and 10 for TurboDB for .NET.

3. Then recompile your application.

To activate your TurboDB license:

The license of an evaluation edition allows you to use all features of TurboDB without restrictions. However, **applications compiled with the evaluation edition run only for three days**. After this period they will stop working and throw an exception indicating that the trial period is over. The TurboDB evaluation installation itself will never expire, therefore you just have to **rebuild** your application again and it will run for another three days. If you can live with the fact, that your application has to be re-built every three days, you will never have to buy a license.

When you buy a real license, you will receive a different license file that you must copy into the directory of the TurboDB assemblies thereby replacing the existing evaluation license file. Then **rebuild** your application and it will stop to expire.

1.3 Licensing TurboDB Managed

TurboDB Managed 2 uses a license that is directly embedded in your application as a resource file.

To include license information for TurboDB Managed in your application:

1. Add the file *TurboDBManagedLicense.resx* to your application.
2. Recompile your application.

Note: Due to a bug in .NET Compact Framework 2.0, the license mechanism does not work with the original version of Compact Framework. The application always throws an exception saying that TurboDB Managed was not licensed. You must at least install SP 1 to make the license work.

To activate your TurboDB license:

A valid trial license which runs 30 days from installation can be found in the TurboDB Managed installation directory. When you buy a production license you will receive another license file, that you can rename and store as you like. Just remove the prior license file from the project and add the new one.

1.4 Sample Programs

Sample programs for TurboDB Managed are installed in the *MyDocuments* folder of the user, who has installed TurboDB.

Sample programs for TurboDB Win32 for ADO.NET are installed in the installation folder, which is by default *<program files>\dataweb\TurboDB for .NET*.

1.5 Support

dataweb provides technical support for TurboDB in the Internet.

FAQ

Is your question in the frequently asked questions on the Web? Have a look at http://www.dataweb.de/en/support/faq_general.html.

Web Site

Visit the TurboDB Web site at http://www.turbodb.de.

Forum

dataweb offers a forum called TurboDB Diskussionsforum, which is available in English and German. Another forum in German language for TurboDB and related issues (i.e. TurboDB Studio) is to be found at http://www.tdb.de/.

E-Mail

If you have questions or problems that are not answered on the Web Site or in the forum then send a mail to dataweb support team. Again, English and German messages are supported.

Professional Services

are provided by our service partners. In Germany, this is pohmann & partner found at http://www.pohmannundpartner.de.

1.6 Database Engine

dataweb currently offers two database engines known as TurboDB Win32, which runs on all 32-bit Windows platforms and TurboDB Managed, which runs on .NET Framework and .NET Compact Framework.

The TurboDB Engines are a very fast and compact database kernel that have proven to fit the needs of application programmers over the last eight years. They do not need any configuration, so they can be installed by just copying the program files. This feature makes them an ideal solution for downloadable, CD, DVD, mobile or Web applications, that are to run on a remote Web server only available via ftp.

This book contains all the features of the TurboDB database engines themselves, i.e. features independent of the development environment and the component library used. For information specific to the VCL component library for Delphi and C++ Builder, please refer to "TurboDB Win32 for VCL". The classes of the ADO.NET provider for the .NET framework are described in "TurboDB Win32 for ADO.NET" and "TurboDB Managed".

Both TurboDB database engines have the following advantages:

- Small footprint
- Fast
- Runs without installation
- No configuration needed
- Supports multi-user access
- Royalty-free deployment
- Visual database manager included
- Many additional tools available for free

The **managed engine** has the following additional advantages:

- Runs on Windows for mobile devices: Pocket PC and Smartphone.
- Does not require any special rights for running in a .NET environment (e.g. unsafe code)

- Supports user-defined functions, stored procedures and user-defined aggregates.

The **native engine** has these additional advantages:

- Encrypted database files
- Special column types for one-to-many and many-to-many relationship between tables

1.6.1 New Features and Upgrade

1.6.1.1 New in TurboDB Win32 v5

TurboDB 5 comes with a large number of new and enhanced features. Note that some of those new features require existing tables to be updated to table level 4 in order to be available. Check in the respective section to learn whether this is the case.

- Uncorrelated and correlated sub-queries
- The top keyword restricts the number of rows in the result set
- Strong encryption based on either Blowfish or Rijndael (AES) algorithm
- Time fields with a resolution of seconds or milliseconds
- Default values for table columns
- Checks for table rows
- The CASE construct
- Referential integrity with optional action cascading
- Ultra-fast maintained full-text indexes with ranking and better SQL integration
- Transaction support with automatic crash recovery
- Better typed aggregated columns
- Join conditions can now be arbitrary search-conditions including calculations and logical operands, not just comparisons
- UNION, EXCEPT and INTERSECT
- Nullable character types
- Managed databases with database-wide properties
- The CAST function
- Optimized page sizes according to predefined capacity (table level 4)
- Supports up to 50 indexes per table (table level 4)
- The index files are named <table name>|<index name>.ind to prevent naming conflicts (table level 4 only).
- Supports column names with spaces and special characters (table level 4)

1.6.1.2 Upgrade TurboDB Win32 v5

When upgrading from TurboDB 4 to TurboDB 5 some points have to be considered:

Important Hint

Please note, that TurboDB 5 cannot share TurboDB databases with TurboDB 4 at the same time. You will receive an error message "Error in log-in". If there is no TurboDB 4 application currently running, but there are still net/rnt/mov/rmv files around, the same error will shine up. Delete the files and everything will work as expected.

New Reserved Keywords

The following identifiers are new reserved keywords in TurboDB 5. If your database schema is using those words as table or column names, you must either enclose them in double quotes wherever they appear in SQL statements or use different names:

ACTION, ALL, ANY, CASCADE, CASE, CAST, DICTIONARY, ENCRYPTION, EXCEPT, EXISTS, FULLTEXTINDEX, INTERSECT, NO, SOME, TOP, UNION, WHEN

Use of Quotes

TurboDB 5 uses single-quotes to denote string literals exclusively. Double quotes were allowed in TurboDB 4 as well but must be changed when upgrading. Double quotes now exclusively denote identifiers and can be used to work with names that contain spaces or are identical to reserved keywords.

Encryption

TurboDB now supports additional methods for strong encryption. Due to this fact the syntax of the *create table* statement and the *alter table* statement has been modified.

Full-Text Indexes

Full-text indexes have been thoroughly re-designed to be faster and maintained. The old full-text indexes are still supported for the older table levels but if you want to upgrade your tables you have to re-write the full-text part of your SQL code.

1.6.1.3 New in TurboDB Managed v2

Version 2.0

- The new CONTAINS predicate together with the new full-text indexes allow for powerful and very fast search for arbitrary words.
- Table names can have up to 79 characters and column names up to 128 characters.
- Connections are now pooled for better performance in applications that execute multiple commands and close the connection in between.
- Fast encryption and Rijndael (AES) encryption is now supported.
- Programmatic compression of databases is provided by the *TurboDBConnection* class.
- UNION, INTERSECT and EXCEPT statements are implemented.
- TurboDB Pilot has a much more comfortable management of databases and commands.
- TurboDB Pilot now provides a visual table designer for creating and altering database tables.

Version 1.3

TurboDB Managed 1.3 contains a completely new family of features centered around programmability.

- User-defined functions allow you to define your own SQL functions either in SQL or as a .NET assembly.
- Stored procedures allow you to call complex statement sequences with a single call. They are implemented either in TurboSQL or in any .NET language.
- User-defined aggregates allow you to define new aggregation function for your SQL statements in any .NET language.

Read the section on "Programming Language" for detailed information.

1.6.1.4 Upgrade to TurboDB Managed v2

TurboDB Managed 2.0 is fully compatible with version 1.x.

New reserved keywords

There are however some new keywords. If these identifiers are used as table, column or routine names, they have to be quoted with brackets [...] or double quotes "..." in statements: BY, CORRESPONDING, ENCRYPTION, EXCEPT, INTERSECT, UNION, CONTAINS, FULLTEXTINDEX.

1.6.2 TurboDB Engine Concepts

This chapter explains basic concepts of the TurboDB database engines.

Overview describes general features like limits and naming.

Databases explains the difference between single-file databases and directory databases.

Indexes explains the different kinds of indexes TurboDB supports.

Automatic Linking presents the TurboDB concept for quicker and less error-prone data modeling.

Multi-User Access and Locking describes, how TurboDB implements the multi-user access.

Optimization lists items you can check, when your application is not fast enough.

Error Handling describes, how errors from the database are reported and handled.

Miscellaneous talks about the physical database files, data security and localization.

1.6.2.1 Overview

Limits

Table and Column Names

Column Data Types

1.6.2.1.1 Compatibility

There exist two implementations of the TurboDB database engine, the Win32 engine and the .NET engine also called the managed engine. The two database engines can work with the same database files as long as the restrictions of the two engines are observed. These are the rules to follow, when you want to share a database file between TurboDB Managed 2.x and TurboDB for Win32 5.x.

- Use a single-file database instead of a directory database.
- Restrict table and column names to 40 characters.
- Do not use Blowfish encryption, but employ *FastEncrypt* or *Rijndael*. The password for all tables must be the same as TurboDB Managed only supports one password per database.
- TurboDB for Win32 will ignore user-defined functions and stored procedures. They cannot be used in computed indexes and queries.
- Do not use language drivers.
- TurboDB Managed and TurboDB for Win32 cannot share the same database file concurrently since the locking system is different. They can however access the data alternately.

1.6.2.1.2 Limits

Some technical data valid for level 4 and 5 tables:

Maximum number of records per table	2 G
Maximum size of a data table	4 EB (one exabyte is 1 G times 1 GB)
Maximum number of columns per table	1000
Maximum size of one table row	32 KB
Maximum number of user-defined indexes per table	48 (table level 4 and above) or 14 (up to table level 3)
Maximum number of levels in a hierarchical index	255
Maximum size of the calculated index information	32 KB
Maximum number of tables per database	255
Maximum length of a string column	32.767 characters
Maximum number of link columns per table	9
Maximum total size of all blobs of a table	1 TB (block size 512 B) to 64 TB (block size 32 KB)

1.6.2.1.3 Table and Column Names

Identifiers

Identifiers consist of a true character followed by alphanumeric characters, the underscore _ and the hyphen. They can contain up to 40 characters. German umlauts count as true characters.

Valid identifiers are:

```
Street
Avarage_Duration
Date-of-birth
Address8
Lösung
```

Column and Table Names

Column and table names can contain any ANSI characters but the control characters, the brackets [] and the double quotes. If the name follows the rules for an identifier, it can be used normally in all expressions and statements. If it does not, it must be included either in double quotes or in brackets.

These are examples of **invalid** identifiers, which can be used as column names if enclosed in double quotes or brackets:

`No of Items` (spaces not allowed)
`3645` (first character must be non-digit)

The maximum length for column and table names is 40 characters for table levels up to 4. Table level 5 allows table names up to 79 characters and column names up 128 characters.

1.6.2.1.4 Column Data Types

Turbo Database offers the following types of table columns:

String

A string field holds alphanumeric characters up to the given limit. The maximum size is 32765 characters (255 for table level 3 and below). A string field holds one byte for each character plus one or two bytes for the length of the string plus an additional byte, if the string is nullable.

WideString

Up to 16382 Unicode characters (255 for table level 3 and below). The actual field size in bytes is twice the number of characters plus two for the string length plus one, if the string is nullable.

Byte

Numbers from 0 to 255. Byte fields can have an optional null indicator. The size is one or two bytes depending on the null indicator.

SmallInt

Numbers from -32767 to +32768. SmallInt fields can have an optional null indicator. The size is two or three bytes.

Integer

Numbers from -2147483648 to +2147483647. Integer fields can have an optional null indicator. It takes four or five bytes in the database table.

LargeInt

Numbers from –2^63 to +2^63 – 1 with an optional null indicator. One Int64 field uses eight or nine bytes in the table.

Float

Holds a 8 byte floating number, i.e. from 5.0e-324 to 1.7 x 10e308. Can have an optional null indicator. The size is eight or nine bytes.

Time

Values from 12:00:00.000 am to 11:59:59.999 pm. Precision is either minutes, seconds or milliseconds and must be given when creating a level 4 table and above. In level 3 tables and below, precision is always minutes. Can have an optional null indicator. Depending on precision and null indicator, size is between two and five bytes.

Date

Values from 1/1/0000 to 12/31/9999. Size is four bytes. Internally dates are represented as a packed bit field.

DateTime

Values from 1/1/0000 12:00:00.000 am to 12/31/9999 11:59:59.999 pm. Values take eight bytes internally.

Boolean

Holds a Boolean value: *True* or *False*. Can have an optional null indicator. Size is one or two bytes.

Enum

Holds one of a definable set of named values, e.g. mon, tue, wed, thu, fri, sat, sun. The values have to be valid identifiers similar to column names. They can be converted to textual representation using the function *Str*. Example for a enumeration field *gender* with values male, female, unknown: When the value female has been assigned *Str(gender)* returns the string 'female' while *gender* by itself returns the number 2.
An enumeration value may have up to 20 characters, the maximum number of enumeration values is 15 and the total length of all enumeration values including separators must not exceed 255 characters.

Memo

Long strings of variable length up to 1 GB. Memos are stored in additional storage objects called the memo file (extension mmo).

WideMemo

Unicode string of variable length up to 1 GB. Wide memos are stored in the blob storage object (extension blb).

Blob

Images and other binary data of variable length up to 1 GB. Blobs are stored in an additional storage object (extension blb).

Link

Pointer to another record in the same or another table (1:n relation). The target table is fixed for all link values of this column. Links are explained in "Automatic Linking". A link field contains the record id of the record the field is linked to. Its size is four bytes.

Relation

Pointer list to other records in the same or another table (m:n relation). The target table is fixed for all link values of this column. Relations are explained in "Automatic Linking". Relations fields are not physical columns in the database table itself. There is an additional database table created transparently for each relation field that holds one link per row. The relation table has two link columns, one of which points to the database table that has the relation field and the other one points to the database table the relation field links to. Such the relation field itself has a size of zero bytes.
Feature is not supported in TurboDB Managed

AutoInc

Counter that gives a unique number to each record. AutoInc fields are assigned its values by the database engine and can not be edited. The Automatic Data Link mechanism uses AutoInc fields as the primary index to store record references. An AutoInc column has an optional *indication* property, which can be set to a list of column names. This property is used for Automatic Linking and can be left empty to make the *AutoInc* column behave just like a "normal" one.

GUID

128 bit number used by MS COM and ActiveX technologies. GUID stands for Globally Unique Identifier. GUIDs are usually calculated by a OS function which assures that no other call anywhere on earth will produce the same value.

1.6.2.2 Databases

TurboDB supports two different types of databases.

Single-File vs. Directory Database

Directory Database

Directory databases are folders on a hard disk where all TurboDB database objects reside in different files. Database directories have been supported ever since TurboDB exists.

Single-File Database

A single-file database is one single file which contains all database objects of the database. Such a database file has the default extension of *.tdbd. Single-file databases are supported as of version 4.0. Single-file databases have the advantage of being very easy to deploy or just to copy and move around on your hard disk. The advantage of database directories is, that they are slightly faster and that you can share tables between different databases.

Single-file databases are implemented using a virtual file system layer by dataweb. This layer maps the database objects either to different files in a database directory or to a single database file. dataweb offers a tool - the dataweb Compound File Explorer - which can open such database files and show the content. You may also move database objects from and to the database file. This is a way to convert a directory-based database to a single-file database or vice versa as well.

While TurboDB Native supports both of these database types, TurboDB Managed can only work with single-file databases.

Databases with Catalogs

In previous versions TurboDB databases were just a collection of database tables stored in separate files. While this approach has the advantage of being able to share database tables between databases, it also has some disadvantages. One disadvantage is, that a table name cannot always be resolved, because the table file may be located in a folder far away. Another one is, that multiple passwords are required to open the database, if the tables are encrypted in different ways.

For this reason TurboDB 5 for Win32 introduced databases with catalogs, which store a list of tables and additional database-wide properties. Catalogs are most useful for dictionary databases. With singe-file databases, the above disadvantages do either not exist or are less problematic.

TurboDB Managed supports only single-file databases and keeps track of all necessary information by itself. It does therefore not offer an explicit catalog support.

Note: Databases with catalogs were called managed databases in previous versions but the term conflicts with TurboDB Managed .

1.6.2.2.1 Sessions and Threads

As of TurboDB version 4 you need to create a session before you can open tables and queries. If you are using a component library however (e.g. TurboDB for VCL or .NET) session handles are hidden within a database or connection object.

You may create as many sessions as you want, but you should be aware of some consequences in a multi-session application:

- Cursors of the same table within different sessions are synchronized on file level. This is much slower than the synchronization of cursors within the same session, which is performed in the memory.
- You can use different threads for different sessions, but you should not use different threads on the same session. For performance reasons there is no built-in thread synchronization within the same session.

1.6.2.2.2 Table Levels

Along with the enhancements and improvements in TurboDB, different storage formats have been developed to support additional features. They are called table levels and the following will describe the characteristics of each. While TurboDB Native supports all these table levels, TurboDB Managed can only work with table level 5 and above.

Table Level 5

- Is compatible with TurboDB Managed.
- Supports standard SQL syntax for checks, default values and calculations for columns and indexes.
- Supports tables with the same name in different database files in the same directory.
- Includes the table schema in the encryption.
- Is prepared for character set support.
- Is prepared for Unicode table and column names.

Table Level 4

- Supports primary keys and unique keys.
- Supports time columns with a precision of seconds or milliseconds.
- Supports additional (strong) encryption algorithms.
- Supports checks and foreign keys.
- Adjusts the ordering to SQL standard, such that null values are less than all other values.
- Increases the number of indexes allowed for a table.
- Supports strong encryption.
- Allows for maintained full-text indexes.

Table Level 3

- Adds Unicode strings, date time columns and GUID columns.

Table Level 2

- Introduces 32 bit Integers and Ansi encoded strings.

Table Level 1

- The original table file format. Compatible with TurboDB for DOS.

1.6.2.3 Indexes

Indexes are additional storage objects for a database table that enable fast searching and sorting. TurboDB indexes are built on either a list of field names or an expression to define the sorting order. If an index is declared to be unique, records that would create a duplicate key in the index are not accepted. Another kind of indexes are full-text indexes.

Indexes Based on a Field List

These indexes are sorted in the order of the first field in the field list. If two records have the same value for the first field they are sorted after the second field of the field list and so on. There can be up to 10 fields in the index field list. Every field can be sorted in ascending or in descending order.

Indexes Based on an Expression

These indexes are sorted after the value of an arbitrary expression that can be up to 40 characters long. If the expression is of string type, the index is sorted like if the expression values were values of a string column. If the expression is of numeric type, the index is sorted according to normal numeric order.

Full-Text Indexes

A full-text index enables the user to search for a keyword or a set of keywords in any field of the table. Full-text indexes require a separate table, the dictionary table, which contains the indexed words. Full-text indexes are implemented differently for table level 4 and the levels below. In table level 4, there is only one storage object to make-up the connection between the dictionary and the table, it has the extension *fti*. In the older table levels, the connection was implemented using relation fields, which requires an additional base table (extension rel) and two indexes (extension in1 and in2).

Indexes can be created and deleted with various TurboDB tools at design time. At run-time, you can use TurboSQL to create, update and delete indexes. Also some component libraries (e.g. VCL) contain methods for adding and deleting indexes.

1.6.2.4 Automatic Linking

Very often tables are linked the same way in all queries. E.g. items are linked to the invoice they belong to, authors are linked to the books they have written and so on. Therefore TurboDB allows you to specify different links from one table to other tables in the table itself.

Imagine you have an invoice table where the records contain the date of the invoice, the customer no, the invoice no and other invoice-related information. The items are in another table that has columns like *article no*, *price*, *total amount* and others. How do you link the item to the corresponding invoice it is part of? The traditional way is to have an additional column in the item table that designates the invoice no of the invoice the item belongs to. Every query that respects the invoice-item relation has to contain the following condition: ...where "ITEM.invoice no" = INVOICE.no...

What is it?

Even if you can still do this the traditional way with TurboDB, the preferred way of doing it is a little different. Rather than having an *invoice no* in the ITEM table you would use a pointer to the INVOICE table called link field. Because the default in TurboDB is to have a (unique) record id in every table the link column in the item table just holds the record id of the invoice it belongs to. Because the definition of the link column contains the information that the values in this column

point to table INVOICE, the database now knows about this relation and will by default assume it in every query. This way of linking tables has some great advantages:

Doing queries with linked tables is easy because the system "knows" how the tables have to be linked. Queries can be much faster, because a record id is just a number while secondary keys often are much more complex. Changing indexes, column names or types does not affect the link at all. It is very easy to access the record of the master table with a special link notation.

You can look at link fields as an object-oriented way to work with database tables. They do not strictly conform to the relational paradigm but bring the feeling of pointers and references into the game. The item "knows" to which invoice it belongs. This link is given by the nature of things and will not probably change very often.

Another advantage is that link and relation fields need not display the purely technical AutoInc values to the user. If you assign an indication to the AutoInc column, link and relation columns will display this information instead of the numerical one. Here is an example:

CREATE TABLE DEPARTMENT (Name CHAR(20), Id AUTOINC(Name)

CREATE TABLE EMPLOYEE (LastName CHAR(40), Department LINK(DEPARTMENT))

The query

SELECT * FROM EMPLOYEE

will display a list of last names and department names, because the department name is defined as the indication for the AutoInc column.

How is it done?

While link columns introduce easy 1:n relations (one invoice has many items), this object-oriented concept makes as ask for a m:n relation i.e. a list of pointers in one table pointing to another table. TurboDB relation fields are the answer to this. A table containing a relation field to another table links every record to a number of records in the other table and vice versa. Taking again books and authors as example, inserting a relation column in the BOOK table would take care of the fact that a book can be written be more than one author and that one author might contribute to more than one book.

As you might suspect, relation fields are not so easy to implement as link fields. M:n relations have to be realized by an additional table in between that has one record for every link between the tables. This is exactly what Turbo Database does when you define a relation column for your table pointing for example to table AUTHOR. TurboDB will create a hidden intermediate table containing a link column to table AUTHOR and another link column to table BOOK. This is what you had to do if you worked in the traditional way. But with TurboDB the intermediate table is created and maintained automatically and transparently to you.

Compatibility Information

This feature is only partly supported in TurboDB Managed. TurboDB Managed currently supports link columns but not relation columns.

1.6.2.4.1 Working with Link and Relation Fields

In order to profit from automatic linking you should think of adding link and relations columns to every table you create. You will soon find it very natural to add the linking information into the table. After all you do the same with your Delphi, C++ and/or Java classes, don't you?

Adding Link and Relation Columns

When you want to work with link and relations fields to establish a one-to-many or many-to-many relationship between tables, you must decide which of the tables is the source and which is the target of the relationship. The first is called the child table and the second the parent table. It is just like the referencing table and the referenced table when you are working with traditional foreign keys.

The parent table must include an AutoInc column, which is used at the primary key for the linking.

The child table must include a link or relation column to establish the relationship. The link column can store exactly one pointer to the parent table. The pointer is displayed as the AutoInc value in the parent table or as the column values of the indication, if you have defined one with the AutoInc column of the parent table. The relation column stores multiple pointers to the parent table, which are displayed as a list of AutoInc values or column values according to the indication definition for the AutoInc column.

Link and Relation Columns with Direct Table Access

(Direct table access is a feature available for the VCL component library but not with ADO.NET.)

Once you have defined your links and relations, they are respected by the database in every query. Even if you don't have any search-criteria, only corresponding detail records will be shown for every master record. In the rare case that you don't want this default linking you may always enter another equate join that overrides it.

Link and Relation Columns with TurboSQL

In TurboSQL queries the relationships defined through link and relation fields are not created automatically. Use a simple JOIN to utilize the reference:

```
SELECT * FROM Master JOIN Detail ON Detail.LinkField = Master.RecordId
```

For adding new rows to the tables, the function *CurrentRecordId* should be used:

```
INSERT INTO Master VALUES(...); INSERT INTO Detail VALUES(...,
CurrentRecordId(Master), ...)
```

This compound statement inserts first a record in the Master table and then a record into the Detail table while using the last value for the record id in the master table as the new value for the link field in the detail table. Therefore the detail record is linked to the master record.

Compatibility Information

This feature is only partly supported in TurboDB Managed. TurboDB Managed currently supports link columns but not relation columns.

1.6.2.5 Transactions

TurboDB supports transactions based on an additional storage object per table, the redo-log. When an transaction is started, all subsequent modification to the database tables will be entered in the redo-logs after they have been materialized to disk. If the transaction is committed, the redo-log is simply deleted. If the transaction is rolled-back, the information from the redo-log is used to undo the modifications. This means, TurboDB follows an optimistic transaction schema: Committing a transaction is very fast, while undoing it, requires much more work.

The tables, which have been modified by the transaction are locked to other sessions until the transaction is finished. For performance reasons, tables, which have been read during the transaction are not locked. Therefore the transaction level in TurboDB is read-committed.

Because TurboDB clients can interact on file level (i.e. without a database server), the handling of clients that die during a transaction is more difficult. TurboDB engine can detect the fact, that another client has died and performs the roll-back. Because in the scenario, another client undoes the modifications of the died client, we call this mechanism hijacking.

1.6.2.6 Optimization

When you have the feeling that your TurboDB application is slower than it should be, there are many possible reasons. Check the following questions and follow the appropriate instructions.

One or more select statements on a large table or on a set of tables takes too long to execute

There are basically two ways to speed a query: Create the necessary indexes and/or optimize the statement.

Setting a filter on a table component (VCL library) takes too long

Adding an index to your table may also help in this case. Another way is to use the range feature instead of the filter.

In file access mode local operation is quite fast, but as soon as a second application just connects to the database, everything slows down

The first thing to check in this case, is whether your network has problems. Because file access mode makes use of network functionality very heavily, it often happens that poor network performance was not noticed before the TurboDB application was installed.

The network is ok, but when a lot of people edit in the database, completing an operation takes very long

With a lot of people working concurrently on the database, the locking overhead grows and a lot of waiting for access to the database occurs. The first step in this case is to use explicit locking to form bigger database operations and thus less locking overhead. If this won't help enough, you can still use TurboDB Server.

In addition to those specific hints, there some general hints that can help increase database performance:

Set the flush mode to fast

The flush mode determines the buffering degree within the database. Setting it to fast will maximize the internal buffering and therefore increase performance. However, if the application crashes, data loss can occur in this mode.

Use exclusive access if possible

In case your application is conceived for a single-user only. You should put the database in exclusive mode to eliminate multi-user access overhead.

1.6.2.6.1 Network Througput and Latency

The network performance is critical for the performance of your TurboDB applications especially, when it is used in multi-user mode. Network problems are the most frequent reason for poor database throughput and should therefore checked out first in case of performance problems. You might have a network problem, if the computer with the database files is different from the one with the application and your application is generally very slow or gets generally very slow, as soon as a second database client is started.

Because network problems are sometimes hard to detect, dataweb offers a free program, which measures network operations typical for databases called *NetTest*. You can download this program from our Web site or request it from the dataweb support. This test program will enable you to determine within five minutes, whether the network is the cause of a performance problem or not.

In case the network turns out to be slow, there are different points to check out:

- Check whether some virus checker or other software is conflicting with TurboDB files. These programs tend to check these highly dynamic files after each modification and thereby slow down or even completely block the database access. You can safely configure the virus checker to not check TurboDB files, because they are not executable.
- Check whether there is updated driver software for your network adapter or if it is defect.
- There is a known problem with SMB signing, if you access a Windows 2000 domain controller from Windows XP clients. Please refer to Microsoft knowledge base article 321098 for more information and the resolution.
- Hubs between database clients and the database file server sometimes block access, if the network traffic is too high. In this case they should be replaced by a good switch.

1.6.2.6.2 Secondary Indexes

Additional indexes for database tables can accelerate queries and filters by some orders of magnitude. Consider a simple query like:

```
select * from Customers where State = 'NJ'
```

or the similar filter condition

```
State = 'NJ'
```

Without an index, TurboDB has to scan every record in the table to select those, which satisfy the condition. And while TurboDB is optimizing multiple reads quite well, the operation nevertheless can take up to some minutes, if the table is large (a few million records).

If however there is an index, which starts with the State column, calculation the result of this selection is instantaneous, because TurboDB will be able to directly sort out the correct rows.

Also with joins, an additional index can do wonders. Look a this query:

```
select * from Customers, Orders where Orders.CustNo = Customers.No
```

or the equivalent

```
select * from Customers join Orders on Orders.CustNo = Customers.No
```

Also in this case, an index over *Orders.CustNo* or *Customers.No* will speed the query considerably. Depending on which one exists, TurboDB will execute the statement such that it can be used. However, since *Orders* will be a much larger table than *Customers* (the average number or orders per customer should be greater than one), an index over the *Orders.CustNo* field will bring you more in terms of execution time gain than an index over *Customers.No*. (The latter will most probably exist nevertheless, because *No* tends to be the primary key for the *Customers* table.)

The disadvantage with indexes is that their maintenance takes time during the change operations delete, insert and update, which must be taken into account as well, when regarding the overall performance of your application. Because in most applications queries are much more frequent than changes, indexes for crucial use cases will pay off most of the time.

1.6.2.6.3 TurboSQL Statements

Some rules for fast TurboSQL statement execution:

Start the where and the having clause with simple and-ed conditions

If logically applicable, order your search-conditions like this:

```
A.a = B.a and C.b > X and (...)
```

i.e. start with simple comparisons, which are necessary for the whole search-condition to be satisfied. These simple comparisons are most suited for optimization. The optimizer will try to create this structure of the search-condition automatically but may in some cases not be smart enough to do so.

Separate the column-reference from the value in comparisons

If you write

```
A.a > 2 * :ParamValue,
```

this will be optimized more likely than

```
A.a/2 > :ParamValue.
```

The important point here is that the column reference *A.a* stands alone the left side of the comparison.

Prefer *like* over Upper

The condition

```
A.a like 'USA'
```

can be optimized, while

```
Upper(A.a) = 'USA'
```

cannot.

Prefer left outer joins over right outer joins

The implementation of joins largely favors left outer joins. Whenever it is suitable in your application, write

```
B left outer join A on B.a = A.a
```

instead of

```
A right outer join B on A.a = B.a.
```

This can speed up your statement considerably. The optimizer does not do this conversion by itself, because it would deprive you of the possibility to hand-optimize your statement.

Modify the Sequence of Tables in the From Clause

This sequence can have a severe impact on the query performance. If you think, your query is not as fast as it should be, just check out different orderings in the table-reference list.

```
select * from A, B, C
where ...
```

might be much faster than

```
select * from C, B, A
where ...
```

Normally the optimizer will try to order table-references not part of a join in the best way, however sometimes assistance from the programmer is needed.

1.6.2.7 Miscellaneous

Database Files describes, which physical database files exist in TurboDB and what they are good for.

Data Security explains the various methods to secure your data.

Localization outlines the way, how you can adopt your TurboDB application to special locales.

1.6.2.7.1 Database Files

This topic explains the files that make up a TurboDB database. They are named after the file extension used. If you are working with a single-file database, you cannot see these file names directly. But when using the dataweb Compound File Explorer, you will see that the database file contains storage objects (also sometimes called files) with the same name extensions.

dat and rel Files

Contain the database tables, that is the records. Rel files are special database tables created transparently to implement many-to-many relationship. Deploy with your application.

mmo and blb Files

These are the memo and blob files, that exist once for each table that has at least one memo field or at least on blob field. One such file contains all the data of all the memo or blob fields in the table. Deploy with your application.

ind Files

User defined index. Each ind file contains one index. Deploy with your application.

id, inr and in? Files

Automatically generated indexes for tables of level 3 and below. The *inr* file is an index on the *AutoInc* field and the *id* file is an index on its indication. The *in?* indexes (in0, in1 etc) index link columns. Deploy with your application.

net, rnt, mov and rmv Files

These are the lock files and exist for each table open in shared mode. Do not deploy with your application since these files contain only dynamic information. When your application crashes or is reset during debugging these files happen to remain on your harddisk and will lead to error messages like "table is in use by another application". In this case, just connect to the database with TurboDB Viewer or any other TurboDB application, view the corresponding tables and the files will be deleted when you close this application.

tra and rtr Files

These files are the redo log files for transactions. During an transaction, there is one *tra* file (*rtr* for relation tables) for each table modified during the transaction. When the transaction is finished, those files are deleted. If you see those files with your database, when no application is currently accessing it, this means, an application crashed during a transaction. Do not delete the redo logs then, the application that accesses the database tables next, will rollback the interrupted transaction to restore database integrity.

Temporary tables and indexes

Temporary tables have random file names like *jzbgopqw.dat* and the temporary indexes are called appropriately. These files are usually stored in the user's temporary directory. But you can define any other directory using the *PrivateDir* property in one of the library components.

tdbd Files

tdbd stands for TurboDB database. Such a file contains all tables and indexes, which belong to a database, if the database was created as a single-file database. In this case there are no real *dat*, *mmo*, *blb*, *rel*, *id*, *in?* and *ind* files seen in the file system, because the respective data is stored within the tdbd file.

1.6.2.7.2 Data Security

Normally your TurboDB database tables can be opened by any person who has access to the file and who uses a tool that can read TurboDB database files. To prevent people from doing so, you can define a password for your tables. All TurboDB tools respect this password and will not show the content of the table unless the user has entered the correct password.
While this is a very useful feature in many cases, it is not a true protection for your data, because one can still read the content of a database or table file with any binary editor or even a text editor. This is also true if you assigned a password to the table, because the password does not change the way database values are stored. If you want to secure your data from being viewed by unauthorized people, TurboDB Engine offers a variety of encryption algorithms, which encrypt every record when it is written to the file.

The classic TurboDB encryption algorithm is based on a 32-bit key. As you might know, a 32-bit key in our days is not secure enough to do banking or other high security things. But for most applications this level of security is appropriate and a shorter key speeds up database transactions.

If you need strong encryption for your data, you can use one of the strong encryption algorithms offered in TurboDB. With these algorithms in place, your data is secured from anybody, who does not know the key. With the current state of encryption technology, these ciphers cannot be broken even with sophisticated decryption algorithms and computer hardware.

The encryption method can be defined on database level (for managed databases) or on table level. If you define the encryption on database level, you have to define the encryption method and the password only once when creating the database. And the user must enter the password only once for all tables in the database. Therefore, this is the recommended way.

In previous versions of TurboDB, encryption required both a password and a 32-bit number called the code to connect to a table. Current versions only require one string, the password. For compatibility, the former combination of password and code is now merged into one string like this <password>;<code>. For example the password *secret* and the code *-3871* are now entered as

the password *secret;-3871*.

This is a list of all available security options. They are indicated as the encryption method enumeration in the different libraries.

Name	Description	Key	Compatibility
Default	For tables in managed databases: Takes the encryption parameters from the database. Other tables and databases: No encryption	See respective row	-
None	Neither encryption nor protection	-	-
Protection	The table is not encrypted but requires a password to be opened	The password, e.g. 3Huv	All table levels. All versions of TurboDB for Win32 . TurboDB Managed 2.x and above.
Classic	The table is encrypted with fast encryption and has an additional password.	The password and the numeric encryption code separated by a semicolon, e.g. 3Huv;97809878	All table levels. All versions of TurboDB for Win32 . TurboDB Managed 2.x and above.
Fast	The table is encrypted with a very fast 32-Bit cipher. Sufficient for many purposes but not 100% secure.	An alphanumeric password up to 40 characters, e.g. 3Huv	All table levels. All versions of TurboDB for Win32 . TurboDB Managed 2.x and above.
Blowfish	Encryption with the well-known Blowfish algorithm using a 128 bit key.	An alphanumeric password up to 40 characters, e.g. 3Huv	Table level 4 and above. TurboDB for Win32 5 and above. Not yet supported in TurboDB Managed.
Rijndael	Encryption with the well-known Rijndael algorithm using a 128 bit key. Also known as Advanced Encryption Standard (AES).	An alphanumeric password up to 40 characters, e.g. 3Huv	Table level 4 and above. TurboDB for Win32 5 and above. TurboDB Managed 2.x and above.
AES	Same as Rijndael.	Same as Rijndael	Same as Rijndael

1.6.2.7.3 Localization

TurboDB offers an interface for language drivers which can implement localized collate sequences for sorting, searching and indexing. Language drivers are dynamic link libraries. The language driver interface is documented in the TurboDB Language Driver Toolkit which is available on our Web site.

A table is bound to a language driver at the time of creation or alteration. Therefore the tools used to create tables all offer the language selection as a property of the database table.

Users working with the VCL edition of TurboDB can also customize the error messages in the database exceptions. The Delphi source file, which contains those messages (*TdbMessages.pas*) is delivered with TurboDB and can be extended to assign texts of another language to the

predefined constants.

Compatibility Information

This feature is not supported in TurboDB Managed.

1.6.3 TurboPL Guide

TurboDB Engine has a set of native built-in functions. These functions have been used for check constraints, calculated indexes and default values in table levels before 5. Some of them can still be used for compatibility reasons in TurboSQL, but are no more recommended.

- Operators and functions
- Search-Conditions

1.6.3.1 Operators and Functions

- Arithmetic Operators and Functions
- String Operators and Functions
- Date and Time Operators and Functions
- Miscellaneous Operators and Functions

1.6.3.1.1 TurboPL Arithmetic Operators and Functions

These arithmetic operators and functions can be used in TurboPL expressions. They are no more recommended for TurboSQL.

Operators

+	Addition
-	Subtraction
*	Multiplication
/	Real division
div	Integer division
mod	Remainder

Comparisons

<	less
<=	less or equal
=	equal
>=	greater or equal
>	greater
less	less
is	equal
greater	greater
<>	not equal
from...upto	range test
in [...]	set test

Functions

Abs(X: Real): Real	Returns the absolute value of the argument X.
ArcTan(X: Real): Real	Returns the arctangent of a given X.
Cos(X: Real): Real	Returns the cosine of the angle X, in radians.
Exp(X: Real): Real	Returns the value of e raised to the power of X, where e is the base of the natural logarithm.
Frac(X: Real): Real	Returns the fractional part of the argument X.
Int(X: Real): Integer	Returns the integer part of X, that is, X rounded toward zero.
Log(X: Real): Real	Returns the natural log of a real expression.
Round(X: Real; [Scale: Integer]): Real	Returns the value of X rounded to the nearest number with the given scale. Scale can be negative as well.
Sin(X: Real): Real	Returns the sine of the angle in radians.
Sqrt(X: Real): Real	Returns the square root of X.

Compatibility Information

TurboPL is supported only for backward compatibility in tables up to level 4.

1.6.3.1.2 TurboPL String Operators and Functions

These string operators and functions can be used in TurboPL expressions. They are no more recommended for TurboSQL.

Operators

+	concatenation, e.g. EMPLOYEES.FirstName + ' ' + EMPLOYEES.LastName
[]	access to single character, e.g. EMPLOYEES.FirstName[1] + '. ' + EMPLOYEES.LastName

Comparisons

<	less
<=	less or equal
=	equal
>=	greater or equal
>	greater
less	less
equal	equal
greater	greater
<>	not equal
has	case sensitive search of string within another, e.g. 'John Smith' has 'Sm'
like	case insensitive correspondence to mask containing jokers, e.g. 'Smith' like 'sMIth', 'Smith' like 'Sm*', 'Smith' like 'Smit?' (all true)
in [..]	tests, whether an element is contained within the set

Functions

Arguments in brackets are optional.

Asc(C: String): Integer	Returns the ordinal value of the first character in the string.

Chr(N: Integer): String	Returns the character for a specified Unicode value.
Exchange(Source, From, To: String): String	Replaces all occurrences of From in Source by To and returns the modified string.
FillStr(Source, Filler: String; Len: Integer): String	Fills the Source with the Filler up to the given Length and returns the result.
LeftStr(Source: String; Len: Integer): String	Return the left substring of source with given length.
Length(Source: String)	Return the count of characters in Source.
Lower(Source: String): String	Returns the string in lowercase.
LTrim(Source: String): String	Returns Source without any leading white-space.
MemoStr(Memo: MemoField [; Len: Integer]): String	Returns the first Len (default is 255, -1 means all) characters of the content of the memo field in the current record.
NTimes(Source: String; Count: Integer): String	Returns a string that repeats Source Count times.
RealVal(Str: String): Real	Calculates the numeric value of a string expression.
Pos(SubStr, Source: String): Integer	Returns the position of SubStr in Source or 0, if SubStr is not contained in Source.
RightStr(Source: String; Len: Integer)	Returns the Len last characters of Source.
RTrim(Source: String): String	Returns Source without any trailing white-spaces.
Scan(SubStr, Source: String): Integer	Returns the number of occurrences of SubStr in Str.
Str(Num: Real[; Width, Scale: Integer]): String	Returns the alphanumeric representation of Num with given Width and Scale. Width=1 means as needed. The alphanumeric representation of an enumeration value (see column data types) is the name of the value.
Upper(Source: String): String	Returns the string in uppercase.
NewGuid: String	Returns a string denoting a new Globally Unique Identifier.

Compatibility Information

TurboPL is supported only for backward compatibility in tables up to level 4.

1.6.3.1.3 TurboPL Date and Time Operators and Functions

These date and time operators and functions can be used in TurboPL expressions. They are no more recommended for TurboSQL.

Comparison

All numeric comparison operators can be used for time, date and datetime values as well. E.g. Date1 > Date2 if and only if Date1 designates a point in time later then Date2.

Calculations

You can add time spans to dates, subtract time spans from dates and subtract dates from each other to get the time span. The time span is a real number, which indicates the number of days (including a fractional part for the time of day) when calculating with dates and datetimes or the number of minutes (including a fractional part for the seconds and milliseconds) when calculating with times.

If Time1 and Time2 are time values, Date1 and Date2 date values, DateTime1 and DateTime2 datetime variables and TimeSpan1, TimeSpan2 real variables, then the following expressions are meaningful:

```
Time2 - Time1
```

```
Time2 - TimeSpan1
Time1 + TimeSpan2
Date2 - Date1
Date2 - TimeSpan1
Date2 + TimeSpan2
DateTime2 - DateTime1
DateTime2 - TimeSpan1
DateTime2 - TimeSpan2
```

Beyond the numeric operators and functions there are also special date and time functions:

Function	Description
CombineDateTime(ADate: Date; ATime: Time): DateTime	Puts a date and a time together to form a datetime.
Day(ADate: DateTime): Integer	Extracts the day out of a date.
Hour(ADate: DateTime): Integer	Extracts the hour from a time or datetime.
Millisecond(ADate: DateTime): Integer	Extracts the millisecond from a time or datetime.
Minute(ADate: DateTime): Integer	Extracts the minute from a time or datetime.
Month(ADate: DateTime): Integer	Extracts the month out of a date.
Now: Time	Returns the current time.
Second(ADate: DateTime): Integer	Extracts the second from a time or datetime.
Today: Date	Returns the current date
Week(ADate: DateTime): Integer	Returns the number of the calendar week within the year.
WeekDayNo(ADateTime: DateTime): Integer	Returns the day of week as a number between 1 (Monday) and 7 (Sunday)
Year(ADate: DateTime): Integer	Extracts the year out of a date.

Compatibility Information

TurboPL is supported only for backward compatibility in tables up to level 4.

1.6.3.1.4 TurboPL Miscellaneous Operators and Functions

These miscellaneous operators and functions can be used in TurboPL expressions. They are no more recommended for TurboSQL.

HexStr(Value: Integer [; Digits: Integer])	Returns the hexadecimal representation of a number with at least *Digits* digits.
CurrentRecordId(TableName)	Returns the last used record id of the given table. Using this function, it is possible to enter linked records in multiple tables within one compound statement.

Compatibility Information

TurboPL is supported only for backward compatibility in tables up to level 4.

1.6.3.2 Search-Conditions

- Filter search-conditions
- Full-text search conditions

1.6.3.2.1 Filter Search-Conditions

Filter search-conditions are used, when entering check constraints for a TurboDB table via TurboDB Viewer or via the VCL TTdbTable component. They are very similar to TurboSQL search-conditions with a few exceptions:

- Date, time and number formats are based on the local settings
- * and ? are allowed as jokers as well as % and _
- Sets are written in brackets instead of parenthesis

Examples

(*Name*, *Amount*, *Amount1*, *Amount2* and *Date-of-birth* are table columns.)

```
Name = 'Smith'
Name like 'Smi*'  (deprecated, prefer % in place of *)
Name like 'Smi%'
Name like 'Smit?'  (deprecated, prefer _ in place of ?)
Name like 'Smit_'
Name has 'mit'
LeftStr(Name, 2) = 'Sm'
Length(Name) > 4
Amount = 13546.45
Amount < 13546.46
Amount < 345.67 or Amount > 567.89 (note that the decimal point depends
on your local settings)
Amount1 * 0.3 > Amount2 * 0.8
Amount is not null (includes all records that have a value for Amount)
Date-of-birth = "4/20/1962" (note that the date format depends on your
local settings)
Date-of-birth < "4/20/1962"
Date-of-birth between "4/1/1962" and "4/30/1962"
Year(Date-of-birth) = 1962
Date-of-birth is null  (includes all records that have no value for
Date-of-birth)
```

Rules for Quoting and Escaping

Both single and double quotes are allowed for string constants. If you need a quote within a string, which is terminated with the same kind of quote, duplicate the quote:

```
"My ""quote""" -> My "quote"
'My "quote"' -> My "quote"
'My ''quote''' -> My 'quote'
"My 'quote'" -> My 'quote'
```

If a table column has a name, which is also a keyword, you must precede it by $:

```
Length($Password) > 8
```

TurboDB offers powerful functions and operators for use in search-conditions, e.g. *like*, *between...and...*, *LeftStr*, *Year* and many others. Refer to Operators and Functions for a complete reference. Comparisons can be combined using the logical operators *and*, *or* and *not*.

Compatibility Information

TurboPL is supported only for backward compatibility in tables up to level 4.

1.6.3.2.2 Full-text Search-Conditions

Full-text search-conditions are used with the TurboSQL contains predicate and the VCL *TTdbTable.WordFilter* property. A full-text search-condition is basically a list of keywords, separated by "+", "," or "-". These characters mean:

, or space	both keywords must occur in the record
+ or /	one of the keywords must occur in the record
-	the keyword must not occur in the record

The alternate character (space and slash) are only available as of table level 4. The keyword itself can contain the jokers "?" and "*" to represent any single character or any substring respectively.

Examples

Database	Finds *Database*, *database*, *dataBase*, ...
Database*	Finds *database*, *Databases*, *DatabaseDriver*, ...
Data?ase	Finds *Database*, *dataCase*, ...
Database, Driver	Record must contain the words *Database* and *Driver*
Database Driver	Same as above for table level 4
Database, Driver, ODBC	Record must contain the words *Database*, *Driver* and *ODBC*
Database Driver ODBC	Same as above for table level 4
Database + Driver	Record must contain either the word *Database* or the word *Driver* or both
Database/Driver	Same as above for table level 4
Database + Driver + ODBC	Record must contain either the word *Database* or the word *Driver* or the word *ODBC*
Database/Driver/ODBC	Same as above for table level 4
Database Driver ODBC/OLE	Record must contain the word *Database* and the word *Driver* and either the word *ODBC* or the word *OLE*
-Database	Record must not contain the word *Database*
Database - Driver	Record must contain the word *Database* but not the word *Driver*

Compatibility Information

TurboPL is supported only for backward compatibility in tables up to level 4.

1.6.4 TurboSQL Guide

TurboSQL is a subset of SQL 92 that contains all of minimal SQL as described by the MS ODBC specification and is very similar to Local SQL used with Borland Database Engine.

Conventions

- Table Names
- Column Names

- String Literals
- Date Formats
- Time Formats
- DateTime Formats
- Boolean Literals
- Table Correlation Names
- Column Correlation Names
- Command Parameters
- Embedded Comments

Data Manipulation Language

- Overview
- DELETE Statement
- FROM Clause
- GROUP BY Clause
- INSERT Statement
- ORDER BY Clause
- SELECT Statement
- UPDATE Statement
- WHERE Clause

Data Definition Language

- Overview
- CREATE TABLE Command
- ALTER TABLE Command
- CREATE INDEX Command
- DROP Command
- Column Data Types

Programming Language

- Overview
- CALL Statement
- CREATE FUNCTION Statement
- CREATE PROCEDURE Statement
- CREATE AGGREGATE Statement
- DROP FUNCTION/PROCEDURE/AGGREGATE Statement
- DECLARE Statement
- IF Statement
- SET Statement
- WHILE Statement

- Exchanging Parameters with .NET Assemblies

1.6.4.1 TurboSQL vs. Local SQL

TurboSQL distinguishes itself in some aspects from Borland's Local SQL:

- In TurboSQL you can enter date, time and datetime literals without quotes, the format is dd.mm.yyyy and HH:mm and dd.mm.yyyy_HH:mm:ss.ms
- TurboDB allows you to issue multiple commands within one statement separated by semicolon.
- TurboDB can rename and modify existing table columns in the ALTER TABLE command.

1.6.4.2 Conventions

1.6.4.2.1 Table Names

Like the ANSI standard TurboSQL confines each table name to a single word comprised of alphanumeric characters and the underscore symbol "_"

```
SELECT *
FROM customer
```

TurboSQL supports full file and path specifications in table references. Table references with path or filename extensions must be enclosed in double quotation marks. For example:

```
SELECT *
FROM "parts.dat"
```

```
SELECT *
FROM "c:\sample\parts.dat"
```

If you omit the file extension for a local table name, ".dat" is assumed.

1.6.4.2.2 Column Names

Like the ANSI-standard TurboSQL confines each column name to a single word comprised of alphanumeric characters and the underscore symbol "_". To distinguish similar column names from different tables preface the table name.

```
SELECT Employee_Id
FROM Employee
```

or

```
SELECT Employee.Employee_Id
FROM Employee
```

In addition, TurboSQL can use the German umlauts for column names and table names:

```
SELECT Kürzung
FROM Beiträge
```

For using column names that contain spaces or other special characters and for distinguishing column names from e.g. function names, you can enclose the column name in brackets or double quotes:

```
SELECT "Employee Id", [Employee Id]
FROM [Update]
```

1.6.4.2.3 String Literals

String literals are enclosed in single quotes. To denote a single quote within a string literal, insert two of them. These are samples for valid string literals:

```
'This is a string literal'
```

```
'This is a ''single-quoted'' string literal'
```

```
'This is a "double-quoted" string literal'
```

And these are invalid:

```
'This isn't a valid string literal'
```

Note

Earlier version of TurboDB allowed also double quotes for string literals. For enhanced SQL compatibility, double quotes are now restricted to denote table and column identifiers.

1.6.4.2.4 Date Formats

TurboSQL offers two different notations for date literals. The native format is dd.mm.yyyy. This format is a very logical one and can not be mistaken by the parser for arithmetic calculations. For this reason, it is not necessary to enclose such a date literal in quotation marks. Example:

```
SELECT * FROM orders
WHERE saledate <= 31.12.2001
```

searches for sales on 31 December 2001.

If you prefer to enter the date in the American or international format, i.e. like 12/31/2001 or 2001-12-31 you have to enclose the date in single quotes and precede it with the keyword *DATE*:

```
SELECT * FROM orders
WHERE saledate <= DATE'12/31/2001'
```

or

```
SELECT * FROM orders
WHERE saledate <= DATE'2001-12-31'
```

The German format works as well:

```
SELECT * FROM orders
WHERE saledate <= DATE'31.12.2001'
```

Leading zeros for the month and day fields are optional. If the century is not specified for the year, TurboDB assumes the 20th century for years from 50 to 99 and the 21th century for years from 00 to 49.

You can omit the keyword *DATE* where the type of the string is obvious like in the above examples.

Example

```
SELECT *
FROM orders
WHERE (saledate > 1.1.89) AND (saledate <= 31.12.20)
```

searches for sales between the January 1st 1989 and the December 31 2020.

1.6.4.2.5 Time Formats

TurboSQL offers two different notations for time literals. The native format expects time literals to be in the format HH:mm; where hh are the hours and mm the minutes. TurboSQL uses the 24 hour scale, that is 2:10 is in the early morning (2:10 AM) while 14:10 is in the early afternoon (2:10 PM). The time literal must not be enclosed in quotation marks.

```
INSERT INTO WorkOrder
(ID, StartTime)
```

```
VALUES ('B00120', 22:30)
```

If you prefer, you can enter the time value in the American format *hh:mm:ss am/pm*. In order to do this, you must enclose the time literal in single quotes and proceed it by the keyword *TIME*:

```
INSERT INTO WorkOrder
(ID, StartTime)
VALUES ('B00120', TIME'10:30:00 pm')
```

Where the type of the string is obvious like in the above example, you can omit the keyword *TIME*. An example where you can not omit it is this one:

```
SELECT StartTime - TIME'12:00:00 pm' FROM WorkOrder
```

Note
The first alternative with the native format without enclosing quotes cannot be used with the Delphi component *TTdbQuery*. The VCL parser for SQL commands recognizes the colon as the starting character of a parameter and creates an error message.

1.6.4.2.6 DateTime Formats

The native format for datetime literals is composed of a date literal and a time literal separated by an underscore '_'. This format can be used with and without quotes:

```
SELECT * FROM WorkOrder
WHERE StartTime >= 31.1.2001_14:10:00
```

or

```
SELECT * FROM WorkOrder
WHERE StartTime >= '31.1.2001_14:10:00'
```

The other way to specify a datetime is to proceed it by the keyword TIMESTAMP and enclose it in single quotes. Again, we have three different representations here:

The American timestamp format:

```
SELECT * FROM WorkOrder
WHERE StartTime >= TIMESTAMP'1/31/2001 2:10:00 pm'
```

The international timestamp format:

```
SELECT * FROM WorkOrder
WHERE StartTime >= TIMESTAMP'2001-1-31 14:10:00'
```

The German timestamp format:

```
SELECT * FROM WorkOrder
WHERE StartTime >= TIMESTAMP'31.1.2001 14:10:00'
```

When the nature of the string is obvious like in the above samples, you can omit the keyword *TIMESTAMP*.

Note
The first alternative with the native format without enclosing quotes cannot be used with the Delphi component *TTdbQuery*. The VCL parser for SQL commands recognizes the colon as the starting character of a parameter and creates an error message.

1.6.4.2.7 Boolean Literals

The Boolean literal values *True* and *False* can be written with or without single quotes. Uppercase and lowercase is ignored.

```
SELECT *
FROM transfers
WHERE (paid = 'True') AND NOT (incomplete = FALSE)
```

1.6.4.2.8 Table Correlation Names

Table correlation names are used to explicitly associate a column with the table from which it comes. This is especially useful when multiple columns of the same name appear in the same query, typically in multi-table queries. A table correlation name is defined by following the table reference in the FROM clause of a SELECT query with a unique identifier. This identifier, or table correlation name, can then be used to prefix a column name.

If the table name is not a quoted string, the table name is the default implicit correlation name. An explicit correlation name the same as the table name need not be specified in the FROM clause and the table name can prefix column names in other parts of the statement.

```
SELECT *
FROM "/home/data/transfers.dat" transfers
WHERE transfers.incomplete = False
```

1.6.4.2.9 Column Correlation Names

Use the optional keyword AS to assign a correlation name to a column, aggregated value, or literal. In the statement below, the tokens Sub and Word are column correlation names.

```
SELECT SUBSTRING(company FROM 1 FOR 1) AS sub, Text word
FROM customer
```

1.6.4.2.10 Command Parameters

TurboSQL uses named statement parameters. They are preceded by a colon:

```
INSERT INTO Customers (Name) VALUES(:Name)
```

The parameter name is the identifier excluding the colon, i.e. *Name* in this case. Whenever you refer to a command parameter in one of the API functions or working with a component library, indicate the identifier without the colon.

When working with the ODBC interface, unnamed parameters are supported as well:

```
INSERT INTO Customers (Name) VALUES (?)
```

1.6.4.2.11 Comments

There are two ways to embed comments into your TurboSQL statement comparable to the comments available in C++. Either you may use /* and */ to enclose the comment or you use // to begin a comment that lasts until the end of the current line.

Example

```
/* 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'
```

1.6.4.3 System Tables

TurboDB uses system tables to store management information and to provide the information schema to the user. The following tables correspond to their counterparts in SQL 92

It depends on the management level of the database whether these tables are permanent or temporary. In both cases you can query on them.

sys_UserTables	Lists the user tables of the database.
sys_UserColumns	Lists the visible columns of the user tables.
sys_UserTableCons	Lists the names of all keys, checks and foreign keys of all user tables.

traints	
sys_UserKeyColumns	Lists all columns from user tables that make part of a (primary or candidate) key.
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.

Examples:

```
select ColumnName, DataType from sys_UserColumns where TableName =
'TableA'
```

Displays the columns and their data types for table *TableA*.

```
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
```

Displays the columns of all unique indexes.

1.6.4.4 Data Manipulation Language

TurboSQL includes the following commands, clauses, functions and predicates for the DML:

Statements

DELETE	Deletes one or more rows from a table.
INSERT	Inserts one or more rows into a table.
SELECT	Retrieves data from tables.
UPDATE	Modifies one or more existing rows in a table.

Clauses

FROM	Specifies the tables from which a SELECT statement retrieves data.
GROUP BY	Combines rows with column values in common into single rows.
HAVING	Specifies filtering conditions for a SELECT statement.
ORDER BY	Sorts the rows retrieved by a SELECT statement.
WHERE	Specifies filtering conditions for a SELECT, UPDATE or DELETE statement.
Sub-Queries	Comparisons to the result of a different query with IN, ANY, SOME, ALL and EXISTS.

Functions, Predicates and Operators

General Functions and Operators	Calculations and comparisons with numbers, strings, timestamps etc.
Arithmetic Functions and Operators	Calculations with numbers

String Functions and Operators	Calculations with strings
Date and Time Functions and Operators	Calculations with date and time
Aggregation Functions	Statistics
Miscellaneous Functions and Operators	Calculations that do not fit in one of the other categories
Table operators	Combine two table into a new one
Sub-Queries	Compare rows with the result of another query
Full-Text Search	Searching for arbitrary keywords anywhere in a row.

1.6.4.4.1 DELETE Statement

Deletes one or more rows from a table.

```
DELETE FROM table_reference
[WHERE predicates]
```

Description

Use DELETE to delete one or more rows from an existing table.

```
DELETE FROM "employee.dat"
```

The optional WHERE clause restricts row deletions to a subset of rows in the table. If no WHERE clause is specified, all rows in the table are deleted.

```
DELETE FROM "employee.dat"
WHERE empno > 2300
```

The table reference cannot be passed to the DELETE statement via a parameter.

Important Note:

The DELETE statement without WHERE clause deletes all rows of a table without checking constraints like foreign keys. This is a feature to provide fast table emptying.

1.6.4.4.2 FROM Clause

Specifies the tables from which a SELECT statement retrieves data.

```
FROM table_reference [, table_reference...]
```

Description

Use a FROM clause to specify the table or tables from which a SELECT statement retrieves data. The value for a FROM clause is a comma-separated list of table names. Specified table names must follow TurboSQL naming conventions for tables. The following examples show different ways, how the from clause can look like:

```
SELECT *
FROM "customer.dat"
```

```
SELECT * FROM
customer, orders
```

```
SELECT * FROM
customer JOIN orders ON orders.CustNo = customer.CustNo
```

Applicability

SELECT

1.6.4.4.3 GROUP BY Clause

Combines rows with column values in common into single rows.

```
GROUP BY column_reference [, column reference...]
```

Description

Use a GROUP BY clause to combine rows with the same column values into a single row. The criteria for combining rows is based on the values in the columns specified in the GROUP BY clause. The purpose for using a GROUP BY clause is to combine one or more column values (aggregate) into a single value and provide one or more columns to uniquely identify the aggregated values. A GROUP BY clause can only be used when one or more columns have an aggregate function applied to them.

The value for the GROUP BY clause is a comma-separated list of columns. Each column in this list must meet the following criteria:

- Be in one of the tables specified in the FROM clause of the query.
- Be in the SELECT clause of the query.
- Cannot have an aggregate function applied to it.

When a GROUP BY clause is used, all table columns in the SELECT clause of the query must meet at least one of the following criteria, or it cannot be included in the SELECT clause:

- Be in the GROUP BY clause of the query.
- Be in the subject of an aggregate function.

Literal values in the SELECT clause are not subject to the preceding criteria.

The distinctness of rows is based on the columns in the column list specified. All rows with the same values in these columns are combined into a single row (or logical group). Columns that are the subject of an aggregate function have their values across all rows in the group combined. All columns not the subject of an aggregate function retain their value and serve to distinctly identify the group. For example, in the SELECT statement below, the values in the SALES column are aggregated (totaled) into groups based on distinct values in the COMPANY column. This produces total sales for each company.

```
SELECT company, SUM(sales) AS TOTALSALES
FROM sales1998
GROUP BY company
ORDER BY company
```

A column may be referenced in a GROUP BY clause by a column correlation name, instead of actual column names. The statement below forms groups using the first column, COMPANY, represented by the column correlation name Co.

```
SELECT company AS Co, SUM(sales) AS TOTALSALES
FROM sales1998
GROUP BY Co
ORDER BY 1
```

Notes

- Derived values (calculated fields) cannot be used as the basis for a GROUP BY clause.
- Column references cannot be passed to an GROUP BY clause via parameters.

Applicability

SELECT, when aggregate functions used

1.6.4.4.4 HAVING Clause

Specifies filtering conditions for a SELECT statement.

```
HAVING predicates
```

Description

Use a HAVING clause to limit the rows retrieved by a SELECT statement to a subset of rows where aggregated column values meet the specified criteria. A HAVING clause can only be used in a SELECT statement when:

- The statement also has a GROUP BY clause.
- One or more columns are the subjects of aggregate functions.

The value for a HAVING clause is one or more logical expressions, or predicates, that evaluate to true or false for each aggregate row retrieved from the table. Only those rows where the predicates evaluate to true are retrieved by a SELECT statement. For example, the SELECT statement below retrieves all rows where the total sales for individual total sales exceed $1,000.

```
SELECT company, SUM(sales) AS TOTALSALES
FROM sales1998
GROUP BY company
HAVING (SUM(sales) >= 1000)
ORDER BY company
```

Multiple predicates must be separated by one of the logical operators OR or AND. Each predicate can be negated with the NOT operator. Parentheses can be used to isolate logical comparisons and groups of comparisons to produce different row evaluation criteria.

A SELECT statement can include both a WHERE clause and a HAVING clause. The WHERE clause filters the data to be aggregated, using columns not the subject of aggregate functions. The HAVING clause then further filters the data after the aggregation, using columns that are the subject of aggregate functions. The SELECT query below performs the same operation as that above, but data limited to those rows where the STATE column is "CA".

```
SELECT company, SUM(sales) AS TOTALSALES
FROM sales1998
WHERE (state = 'CA')
GROUP BY company
HAVING (SUM(sales) >= 1000)
ORDER BY company
```

Note

A HAVING clause filters data after the aggregation of a GROUP BY clause. For filtering based on row values prior to aggregation, use a WHERE clause.

Applicability

SELECT with GROUP BY

1.6.4.4.5 INSERT Statement

Adds one or more new rows of data in a table

```
INSERT INTO table_reference
[(columns_list)]
VALUES (update_atoms)
```

Description

Use the INSERT statement to add new rows of data to a table.

Use a table reference in the INTO clause to specify the table to receive the incoming data.

The columns list is a comma-separated list, enclosed in parentheses, of columns in the table and is optional. The VALUES clause is a comma-separated list of update atoms, enclosed in parentheses. If no columns list is specified, incoming update values (update atoms) are stored in fields as they are defined sequentially in the table structure. Update atoms are applied to columns

in the order the update atoms are listed in the VALUES clause. There must also be as many update atoms as there are columns in the table.

```
INSERT INTO "holdings.dat"
VALUES (4094095, "BORL", 5000, 10.500, 2.1.1998)
```

If an explicit columns list is stated, incoming update atoms (in the order they appear in the VALUES clause) are stored in the listed columns (in the order they appear in the columns list). NULL values are stored in any columns that are not in a columns list.

```
INSERT INTO "customer.dat"
(custno, company)
VALUES (9842, 'dataweb GmbH')
```

To add rows to one table from another, omit the VALUES keyword and use a subquery as the source for the new rows.

```
INSERT INTO "customer.dat"
(custno, company)
SELECT custno, company
FROM "oldcustomer.dat"
```

1.6.4.4.6 ORDER BY Clause

Sorts the rows retrieved by a SELECT statement.

```
ORDER BY column_reference [, column_reference...] [ASC|DESC]
```

Description

Use an ORDER BY clause to sort the rows retrieved by a SELECT statement based on the values from one or more columns.

The value for the ORDER BY clause is a comma-separated list of column names. The columns in this list must also be in the SELECT clause of the query statement. Columns in the ORDER BY list can be from one or multiple tables. A number representing the relative position of a column in the SELECT clause may be used in place of a column name. Column correlation names can also be used in an ORDER BY clause columns list.

Use *ASC* (or *ASCENDING*) to force the sort to be in ascending order (smallest to largest), or *DESC* (or *DESCENDING*) for a descending sort order (largest to smallest). When not specified, *ASC* is the implied by default.

The statement below sorts the result set ascending by the year extracted from the *lastinvoicedate* column, then descending by the *state* column, and then ascending by the uppercase conversion of the *company* column.

```
SELECT EXTRACT(YEAR FROM lastinvoicedate) AS YY, state, UPPER(company)
FROM customer
ORDER BY YY DESC, state ASC, 3
```

Column references cannot be passed to an ORDER BY clause via parameters.

Applicability

SELECT

1.6.4.4.7 SELECT Statement

Retrieves data from tables.

```
SELECT [TOP number] [DISTINCT] * | column_list
FROM table_reference
[WHERE predicates]
[ORDER BY order_list]
[GROUP BY group_list]
[HAVING having_condition]
```

Description

Use the SELECT statement to

- Retrieve a single row, or part of a row, from a table, referred to as a singleton select.
- Retrieve multiple rows, or parts of rows, from a table.
- Retrieve related rows, or parts of rows, from a join of two or more tables.

The SELECT clause defines the list of items returned by the SELECT statement. The SELECT clause uses a comma-separated list composed of: table columns, literal values, and column or literal values modified by functions. Literal values in the columns list may be passed to the SELECT statement via parameters. You cannot use parameters to represent column names. Use an asterisk to retrieve values from all columns.

Columns in the column list for the SELECT clause may come from more than one table, but can only come from those tables listed in the FROM clause. See Relational Operators for more information on using the SELECT statement to retrieve data from multiple tables. The FROM clause identifies the table(s) from which data is retrieved.

If TOP is specified in the statement, the number of rows in the subset is limited to the given number. Top is evaluated after all other clauses and therefore refers to the sorted or grouped result set in case the order by clause and/or the group by clause are present.

If the DISTINCT keyword is present, duplicate rows in the result table are suppressed. DISTINCT cannot be used together with GROUP BY. If a SELECT statement contains both GROUP BY and DISTINCT, the DISTINCT keyword is ignored.

The following statement retrieves data for two columns in all rows of a table.

```
SELECT custno, company
FROM orders
```

See also

JOIN, UNION, INTERSECT, EXCEPT

1.6.4.4.8 UPDATE Statement

Modifies one or more existing rows in a table.

```
UPDATE table_reference
SET column_ref = update_atom [, column_ref = update_atom...]
[WHERE predicates]
```

Description

Use the UPDATE statement to modify one or more column values in one or more existing rows in a table.

Use a table reference in the UPDATE clause to specify the table to receive the data changes.

The SET clause is a comma-separated list of update expressions. Each expression is composed of the name of a column, the assignment operator (=), and the update value (update atom) for that column. The update atoms in any one update expression may be literal values, singleton return values from a subquery, or update atoms modified by functions. Subqueries supplying an update atom for an update expression must return a singleton result set (one row) and return only a single column.

```
UPDATE salesinfo
SET taxrate = 0.0825
WHERE (state = 'CA')
```

The optional WHERE clause restricts updates to a subset of rows in the table. If no WHERE clause is specified, all rows in the table are updated using the SET clause update expressions.

See also

INSERT, DELETE

1.6.4.4.9 WHERE Clause

Specifies filtering conditions for a SELECT or UPDATE statement.

```
WHERE predicates
```

Description

Use a WHERE clause to limit the effect of a SELECT or UPDATE statement to a subset of rows in the table. Use of a WHERE clause is optional.

The value for a WHERE clause is one or more logical expressions, or predicates, that evaluate to TRUE or FALSE for each row in the table. Only those rows where the predicates evaluate to TRUE are retrieved by a SELECT statement or modified by an UPDATE statement. For example, the SELECT statement below retrieves all rows where the STATE column contains a value of 'CA'.

```
SELECT company, state
FROM customer
WHERE state = 'CA'
```

Multiple predicates must be separated by one of the logical operators OR or AND. Each predicate can be negated with the NOT operator. Parentheses can be used to isolate logical comparisons and groups of comparisons to produce different row evaluation criteria. For example, the SELECT statement below retrieves all rows where the STATE column contains a value of "CA" and those with a value of "HI".

```
SELECT company, state
FROM customer
WHERE (state = 'CA') OR (state = 'HI')
```

The SELECT statement below retrieves all rows where the SHAPE column is *round* or *square*, but only if the COLOR column also contains *red*. It would not retrieve rows where, for example, the SHAPE is *round* and the COLOR *blue*.

```
SELECT shape, color, cost
FROM objects
WHERE ((shape = 'round') OR (shape = 'square')) AND (color = 'red')
```

But without the parentheses to override the order of precedence of the logical operators, as in the statement that follows, the results are very different. This statement retrieves the rows where the SHAPE is round, regardless of the value in the COLOR column. It also retrieves rows where the SHAPE column is *square,* but only when the COLOR column contains *red*. Unlike the preceding variation of this statement, this one would retrieve rows where the SHAPE is *round* and the COLOR *blue*.

```
SELECT shape, color, cost
FROM objects
WHERE shape = 'round' OR shape = 'square' AND color = 'red'
```

Note

A WHERE clause filters data prior to the aggregation of a GROUP BY clause. For filtering based on aggregated values, use a HAVING clause.

Applicability

SELECT, UPDATE, DELETE

1.6.4.4.10 General Functions and Operators

There is a list of functions and operators that can be used within TurboSQL expressions. This list is composed of a few standard SQL functions and a lot more additional TurboDB functions.

=

Syntax

```
expr1 = expr2
```

Description

Tests for equality.

<

Syntax

```
expr1 < expr2
```

Description

Tests whether expression *expr1* is lower than *expr2*.

<=

Syntax

```
expr1 <= expr2
```

Description

Tests whether expression *expr1* is lower or equal than *expr2*.

>

Syntax

```
expr1 > expr2
```

Description

Tests whether expression *expr1* is greater than *expr2*.

>=

Syntax

```
expr1 >= expr2
```

Description

Tests whether expression *expr1* is greater or equal than *expr2*.

BETWEEN ... AND ...

Syntax

```
expr1 BETWEEN expr2 AND expr3
```

Description

Tests whether expression *expr1* is greater or equal than *expr2* and lower or equal than *expr3*.

IN

Syntax

```
expr IN (expr1, expr2, expr3, ...)
```

Description

Tests whether *expr* is equal to one of the expressions *expr1*, *expr2*, *expr3*, ...

AND

Syntax

```
cond1 AND cond2
```

Description

Tests whether both *cond1* and *cond2* are true.

OR

Syntax

```
cond1 OR cond2
```

Description

Tests whether at least one of *cond1* and *cond2* is true.

NOT

Syntax

```
NOT cond
```

Description

Tests whether *cond* is false.

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
```

Description

The first form of the case operation determines the first expression for which the condition is true. The second one returns the B expression, who's A expression is equal to *expr*.

Samples

```
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)
```

Description

Converts the value to the given type if possible. The cast operation may cut off strings and loose precision of decimal numbers. If the conversion is not possible, CAST raises an error.

Examples

```
CAST(time AS CHAR(10)) --Converts the time in its string representation
CAST(time AS CHAR(3)) --Displays only the first three characters
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
```

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

1.6.4.4.11 Arithmetic Functions and Operators

This is a list of arithmetic functions and operators that can be used in TurboSQL.

+

Syntax

```
value1 + value2
```

Description

Calculates the sum of two numbers.

-

Syntax

```
value1 - value2
```

Description

Calculates the difference of two numbers.

*

Syntax

```
value1 * value2
```

Description

Calculates the product of two numbers.

/

Syntax

```
value1 / value2
```

Description

Calculates the quotient of two numbers.

%

Syntax

```
value1 % value2
```

Description

Calculates the modulo of two integral numbers.

Compatibility Information

This operator is only available in TurboDB Managed.

ARCTAN

Syntax

```
ARCTAN(value)
```

Description

Calculates the arcus tangens of value.

CEILING

Syntax

```
CEILING(value)
```

Description

Calculates the smallest integral number greater than, or equal to, the given value.

Example

```
CEILING(-3.8) --returns -3.0
CEILING(3.8) --returns 4.0
```

COS

Syntax

```
COS(value)
```

Description

Calculates the cosine of value.

DIV

Syntax

```
a div b
```

Description

Integer division

Example

```
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
```

Description

Calculates the exponential of value (to base e)

FLOOR

Syntax

```
FLOOR(:X DOUBLE) RETURNS DOUBLE
```

Description

Calculates the largest integral number less than or equal to the given value.

Example

```
FLOOR(-3.8) --returns -4.0
FLOOR(3.8) --returns 3.0
```

FRAC

Syntax

```
FRAC(:X DOUBLE) RETURNS DOUBLE
```

Description

Calculates the fractional part of real number.

Example

```
FRAC(-3.8) --returns -0.8
FRAC(3.8) --returns 0.8
```

INT

Syntax

```
INT(:X DOUBLE) RETURNS BIGINT
```

Description

Calculates the integral part of a real number as an integer number.

Example

```
INT(-3.8) --returns -3
INT(3.8) --returns 3
```

LOG

Syntax

```
LOG(:X DOUBLE) RETURNS BIGINT
```

Description

Calculates the natural logarithm of a real number.

MOD

Syntax

```
a mod b
```

Description

Remainder of the integer division. *a mod b = a - (a div b) * b* always holds.

Example

```
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
```

Description

Rounds the value to the given number of digits.

Compatibility Information

This function is only available in TurboDB Win32.

Example

```
ROUND(3.141592, 3) --returns 3.142
```

SIN

Syntax

```
SIN(:X DOUBLE) RETURNS DOUBLE
```

Description

Calculates the sine of a value.

SQRT

Syntax

```
SQRT(:X DOUBLE) RETURNS DOUBLE
```

Description

Calculates the square root of a value.

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

1.6.4.4.12 String Operators and Functions

This is a list of string operators and functions that can be used in TurboSQL.

||

Syntax

```
string1 || string2
```

Description

Concatenates the two strings.

ASCII

Syntax

```
ASCII(string)
```

Description

Calculates the code point of the first character in the string. Returns NULL if the string is NULL or empty.

CHAR_LENGTH

Syntax

```
CHAR_LENGTH(string)
```

Description

Calculates the number of characters in the string.

HEXSTR

Syntax

```
HEXSTR(number, width)
```

Description

Calculates an hexadecimal representation of number with at least width characters.

Example

```
HEXSTR(15, 3) --returns '00F'
```

LEFTSTR

Syntax

```
LEFTSTR(string, count)
```

Description

Calculates the *count* left characters of *string*.

LEN

Syntax

```
LEN(string)
```

Description

Same as *CHAR_LENGTH*. Prefer *CHAR_LENGTH* as it is standard SQL.

LIKE

Syntax

```
string1 [NOT] LIKE string2
```

Description

Compares the two strings as defined in standard SQL using the two joker characters % and _.

Examples

```
'Woolfe' LIKE 'Woo%'
Name LIKE '_oolfe'
```

LOWER

Syntax

```
LOWER(string)
```

Description

Returns the string in lower case.

RIGHTSTR

Syntax

```
RIGHTSTR(string, count)
```

Description

Calculates the *count* right characters of *string*.

STR

Syntax

```
STR(number, width, scale, thousand_separator, fill_character,
decimal_separator)
STR(enumeration_column_reference)
```

Description

The first variant calculates a string representation of the number with the given formatting.

The second variant calculates the string representation of the enumeration value.

Example

```
STR(3.14159, 10, 4, ',', '*', '.') --returns ****3.1416
```

SUBSTRING

Syntax

```
SUBSTRING(string FROM start [FOR length])
```

Returns a partial string of length *length* from *string* starting at *start*.

TRIM

Syntax

```
TRIM([kind [char] FROM] string)
```

Description

Returns a string without leading or trailing characters.
Kind is one of LEADING, TRAILING, BOTH. The default for kind is BOTH.
Char is the character that is trimmed away. The default for char is the space.

Examples

All these expressions return 'Carl':

```
TRIM('  Carl      ')
TRIM(LEADING FROM '   Carl')
TRIM(TRAILING FROM 'Carl    ')
TRIM(BOTH 'x' FROM 'xxCarlxx')
```

UPPER

Syntax

```
UPPER(string)
```

Description

Returns the string in upper case.

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

1.6.4.4.13 Date and Time Functions and Operators

This is a list of date and time functions and operators that can be used in TurboSQL.

+

Syntax

```
date + days
timestamp + days
time + minutes
```

Description

Adds a number of days to a date or timestamp. Adds a number of minutes to a time value.

Examples

```
CURRENT_DATE + 1 --Tomorrow's date
CURRENT_TIMESTAMP + 1 --Tomorrow's time exactly like now
CURRENT_TME + 60 --One hour from now
CURRENT_TIME + 0.25 --15 seconds later
```

-

Syntax

```
date - days
date1 - date2
timestamp - days
timestamp1 - timestamp2
time - minutes
time1 - time2
```

Description

Subtracts a number of days from a date or a timestamp. Subtracts a number of minutes from a time value. Calculates the number of days between two dates or timestamps. Calculates the number of minutes between two time values.

Examples

```
CURRENT_DATE - 1 --Yesterday
CURRENT_TIMESTAMP - 1 --24 hours ago
CURRENT_DATE - DATE'1/1/2006' --Number of days since the beginning of
2006
CURRENT_TIME - 60 --One hour ago
CURRENT_TIME - TIME'12:00 pm' --Number of hours since noon (may be
negative)
```

CURRENT_DATE

Syntax

```
CURRENT_DATE
```

Description

Returns the date of the current day according to your system (local time).

CURRENT_TIME

Syntax

```
CURRENT_TIME
```

Description

Returns the time of the current millisecond according to your system (local time).

CURRENT_TIMESTAMP

Syntax

```
CURRENT_TIMESTAMP
```

Description

Returns the timestamp of the current millisecond (i.e. *CURRENT_DATE* and *CURRENT_TIME* together) according to your system (local time).

DATETIMESTR

Syntax

```
DATETIMESTR(TimeStamp, Precision)
```

Description

Calculates a string representation of the time stamp in the current locale. Precision is 2 for minutes, 3 for seconds and 4 for milliseconds.

EXTRACT

Syntax

```
EXTRACT(kind FROM date)
```

Description

Calculates a value from date. *Kind* is one of these:

YEAR	Returns the year.
MONTH	Returns the month.
DAY	Returns the day.
WEEKDAY	Returns the day of the week. 1 for Monday, 2 for Tuesday etc.
WEEKDAYNAME	Returns the name of the day of the week in the current locale.
WEEK	Returns the number of the week in the year according the ISO standard.
HOUR	Returns the hour.
MINUTE	Returns the minute.
SECOND	Returns the second.
MILLISECOND	Returns the millisecond.

Examples

```
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)
```

Description

Returns the date value for the given date.

Example

```
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)
```

Description

Returns the time stamp value for the given datetime.

MAKETIME

Syntax

```
MAKETIME(hour, minute, second, millisecond)
```

Description

Returns the time value for the given time.

TIMESTR

Syntax

```
TIMESTR(time, precision)
```

Description

Calculates a string representation of the time value in the current locale. Precision is 2 for minutes, 3 for seconds and 4 for milliseconds.

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

1.6.4.4.14 Aggregation Functions

This is a list of aggregation functions that can be used in TurboSQL.

AVG

Syntax

```
AVG(column_reference)
```

Description

Calculates the average of the values in the column. The argument must be a numeric type. The result is always a a FLOAT.

COUNT

Syntax

```
COUNT(*|column_reference)
```

Description

Calculates the number of rows in the column. The argument can be of any type. The result is always a BIGINT.

Examples

```
COUNT(*)
COUNT(NAME)
```

MAX

Syntax

```
MAX(column_reference)
```

Description

Calculates the maximum of the values in the column. The argument must be a numeric type or a date/time type. The result is a super-type of the argument type.

MIN

Syntax

```
MIN(column_reference)
```

Description

Calculates the minimum of the values in the column. The argument must be a numeric type or a date/time type. The result is a super-type of the argument type.

STDDEV

Syntax

```
STDDEV(column_reference)
```

Description

Calculates the standard deviation of the values in the columns. The argument must be numeric type. The result is always a FLOAT.

Example

```
SELECT AVG(Value), STDDEV(Value) FROM Values
```

Compatibility Information

This function is only available in TurboDB Managed.

SUM

Syntax

```
SUM(column_reference)
```

Description

Calculates the sum of the values in the column. The argument must be a numeric type. The result is a super-type of the argument type.

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
User-defined Aggregates

1.6.4.4.15 Miscellaneous Functions and Operators

This is a list of miscellaneous functions and operators that can be used in TurboSQL.

CONTAINS

Syntax

```
CONTAINS(full-text-search-expression IN table-name.*)
```

Description

Evaluates to true, if the row satisfies the full-text search-expression.

NEWGUID

Syntax

```
NEWGUID
```

Description

Creates a new Globally Unique Identifier, like for example {2A189230-2041-44A6-87B6-0AFEE240F09E}.

Example

```
INSERT INTO TABLEA ("Guid") VALUES(NEWGUID)
```

CURRENTRECORDID

Syntax

```
CURRENTRECORDID(table_name)
```

Description

Returns the last used record id of the given table. Using this function, it is possible to enter linked records in multiple tables within one compound statement.

NULLIF

Syntax

```
NULLIF(value1, value2)
```

Description

Returns NULL if *value1* and *value2* are equal and *value1* if they are not.

Compatibility Information

This function is only supported in TurboDB Managed.

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

1.6.4.4.16 Table Operators

TurboSQL supports the following operators to combine table rows. They all follow the standard SQL specification:

JOIN

Syntax

```
table_reference [INNER | LEFT OUTER | RIGHT OUTER | OUTER] JOIN
table_reference
```

Samples

```
SELECT * FROM A JOIN B ON A.a = B.a
```

```
SELECT * FROM A LEFT OUTER JOIN B ON A.a = B.a
```

Description

Returns all row pairs of the two table references, for which the join condition holds.

UNION

Syntax

```
table_term UNION [ALL] table_term [CORRESPONDING BY column_list]
```

Samples

```
SELECT * FROM TABLE A UNION SELECT * FROM TABLE B
```

Description

Returns all rows from the two table terms. The result set is unique if all is not specified. The two table terms must have compatible columns.

EXCEPT

Syntax

```
table_term EXCEPT [ALL] table_term CORRESPPONDING [BY column_list]
```

Samples

```
SELECT * FROM TABLE A EXCEPT SELECT * FROM TABLE B
```

Description

Returns all rows from the first table term that do not exist in the second one. The result set is unique if all is not specified. The two table terms must have compatible columns.

INTERSECT

Syntax

```
table_primitive INTERSECT table_primitive CORRESPONDING [BY column_list]
```

Samples

```
SELECT * FROM TABLE A INTERSECT [ALL] SELECT * FROM TABLE B
```

Description

Returns all rows that exist in both the first and the second table term. The result set is unique if all is not specified. The two table terms must have compatible columns.

1.6.4.4.17 Sub-Queries

Search-conditions within SELECT, INSERT and UPDATE statements may contain embedded queries, which can be compared to the main query via one of the following operators. Furthermore, a select expression in parenthesis can be used everywhere an expression is expected. TurboSQL allows for uncorrelated sub-selects as well as for correlated ones.

IN

Checks whether the value of an expression can be found in the result set of the sub-query.

Example

```
select * from SALESINFO
where customerName in (
  select name from CUSTOMER where state = 'CA'
)
```

Selects all sales to customers from California and is basically the same as

```
select * from SALESINFO join CUSTOMER on customerName = name
where state = 'CA'
```

EXISTS

Checks whether the sub-query contains at least one row.

Example

```
select * from SALESINFO
where exists (
  select * from CUSTOMER
  where name = SALESINFO.customerName and state = 'CA'
)
```

Retrieves the same result as the first example. Note however that this time the sub-query contains a column reference to the outer query. This is called correlated sub-query.

ANY/SOME

Checks whether there is at least one row in the result of the sub-query, which satisfies the search-condition.

Example

```
select * from SALESINFO
where amount > any (
  select averageAmount from CUSTOMER
  where name = SALESINFO.customerName
)
```

Retrieves all sales bigger than the average for the respective customer.

ALL

Checks whether the search-condition is satisfied for all rows in the result of the sub-query.

Example

```
select * from SALESINFO
where amount > all (
  select averageAmount from CUSTOMER
  where state = 'CA'
)
```

Retrieves the sales bigger than the average volume for each single customer in California.

Sub-Query as Expression

A select expression in parenthesis can be used as a scalar expression. The type of the scalar expression is the type oft the first column in the result set. The value of the scalar expression is the value of the first column in the first row. If the result set has no column, the expression is invalid. If it has no rows, the result value is NULL.

Examples:

```
select * from [TableB] where C1 like (select C2 from TableB) || '%'
```

```
set A = (select Count(*) from TableA)
```

Compatibility Information

The use of a sub-select as an expression is only available in TurboDB Managed.

See also

WHERE

1.6.4.4.18 Full-Text Search

Full-text search is the search for an arbitrary word in a table row. This kind of search is especially useful for memo and wide memo fields, where searching with conventional operators and functions does not deliver the expected result or takes too long.

Full-text search in TurboDB has two restrictions:

- There must be a full-text index on the table to use full-text searching capabilities.
- One full-text search-condition always refers to a single table. (There can be multiple full-text search-conditions in the same where clause however.)

The basis of a full-text index is the dictionary, which is a normal database table with a certain schema. It holds the information on indexed words, excluded words, word relevance etc. Once the dictionary exists, it can be used for any number of full-text indexes on one or on multiple tables.

As of TurboDB 5, full-text search-conditions are embedded in the WHERE clause of the statement:

```
select * from SOFTWARE join VENDOR on SOFTWARE.VendorId = VENDOR.Id
where VENDOR.Country = 'USA' and (contains('office' in SOFTWARE.*) or
contains('Minneapolis' in VENDOR.*))
```

A simple full-text search condition looks like this:

```
contains('office -microsoft' in SOFTWARE.*)
```

which is true, if any of the fields of the default full-text index of table SOFTWARE contains the word *office* but not the word *microsoft*. If the query refers to only one table, this can also be written as

```
contains('office -microsoft' in *)
```

If the full-text search-expression contains more than one word without the hyphen, TurboDB searches for rows that contain all the given words. Therefore

```
contains('office microsoft' in SOFTWARE.*)
```

will find rows, that contain both the word office and microsoft in any of the fields of the default full-text index of the table.

Words separated by a plus sign are searched for alternatively. The predicate

```
contains('office star + open' in SOFTWARE.*)
```

finds rows containing the word *office* plus either the word *star* or the word *open* (or both).

Notes

Full-text indexes can be created with the CREATE FULLTEXTINDEX TurboSQL statement, with one of the database management tools (like TurboDB Viewer) or with the appropriate functions of the respective TurboDB access components.

The full-text searching technology has changed between TurboDB level 3 and level 4 tables. The new implementation is much faster and allows for maintained full-text indexes as well as for row relevance. It is strongly recommended to use level 4 tables, when working with full-text searching capabilities. The old full-text search will eventually be removed.

1.6.4.5 Data Definition Language

TurboSQL supports these statements and data types as part of the Data Definition Language:

Statements

CREATE TABLE

ALTER TABLE

CREATE INDEX

CREATE FULLTEXTINDEX

UPDATE INDEX/FULLTEXTINDEX

DROP

Data types

TurboSQL data types

1.6.4.5.1 CREATE TABLE Statement

Creates a new table within the current database.

Syntax

```
CREATE TABLE table_reference
[LEVEL level_number]
[ENCRYPTION encryption_algorithm [PASSWORD password]]
[LANGUAGE language]
(column_definition | constraint_definition [, column_definition |
constraint_definition] ...)
```

where a *column_definition* looks like this:

```
column_reference data_type [NOT NULL] [DEFAULT expression]
```

(see column data types for detailed information)

and a *constraint_definition* like this:

```
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]
```

Description

Use the CREATE TABLE command when you want to add a new table to the database.

```
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
)
```

To define a password, a key and a language, add the appropriate keywords:

```
CREATE TABLE MyTable
  ENCRYPTION 'Blowfish' PASSWORD 'u(i,iUklah'
  LANGUAGE 'ENU'
  (Name CHAR(20))
```

The encryption algorithms currently supported are described in "Data Security". If an encryption algorithm is given, the password must be indicated as well.
The level number is used for backward compatibility. If it is omitted the most current table format will be created.

The other clauses have their standard SQL meaning. Note that TurboSQL does not yet support the *set null* and the *set default* actions for the foreign key present in the SQL standard.

Compatibility Information

The LANGUAGE clause is not supported by TurboDB Managed.

See also

Column Data Types

1.6.4.5.2 ALTER TABLE Statement

Modifies columns and column types of an existing table.

Syntax

```
ALTER TABLE table_reference
[LEVEL level_number]
[ENCRYPTION encryption_algorithm]
[PASSWORD password]
[LANGUAGE language]
DROP column_reference |
DROP constraint_name |
ADD column_definition |
ADD constraint_definition |
RENAME column_reference TO column_reference |
MODIFY column_definition
```

Description

The ALTER TABLE command enables you to modify the structure of an existing table. Please find the description for *column_definition* and *constraint_definition* in the topic about the CREATE TABLE statement. There are six different options:

Delete an existing column with DROP:

```
ALTER TABLE Orders DROP Destination
```

The *column_reference* must refer to an existing column. Note that TurboSQL column names are case-sensitive.

Delete an existing constraint with DROP:

```
ALTER TABLE Orders DROP PrimaryKey
```

Add a new column with ADD:

```
ALTER TABLE Orders ADD Date_of_delivery DATE
```

The name of the new column must not exist before.

Add a new constraint with ADD:

```
ALTER TABLE Orders ADD CONSTRAINT RecentDateConstraint CHECK
(Date_of_delivery > 1.1.2000)
```

```
ALTER TABLE Orders ADD FOREIGN KEY (Customer) REFERENCES Customer
(CustNo)
```

Modify the name of an existing column with RENAME:

```
ALTER TABLE Orders RENAME Date_of_delivery TO DateOfDelivery
```

The first *column_reference* is the name of an existing column, the second is the new name of this column. Renaming a column keeps the data within the column intact.

Modify the column type of an existing column with MODIFY:

```
ALTER TABLE Orders MODIFY DateOfDelivery TIMESTAMP
```

The *column_reference* must refer to an existing column. You may change the column type to any one of the available column types. The column data is kept as far as possible.

The parameters *level_number*, password, key and language have the same meaning as in the CREATE TABLE statement. If password and key are omitted, the current settings are kept. To remove the encryption, set it to NONE.

This statement removes encryption from a table:

```
ALTER TABLE Orders ENCRYPTION None
```

Note: If the password or the encryption mode are to be changed, both the password and the encryption must be indicated for security reasons.

It is possible to combine multiple changes in any order within one single command:

```
ALTER TABLE Orders
  ADD Date_of_delivery DATE,
  DROP Destination,
  ADD DeliveryAddress CHAR(200),
  RENAME Customer TO CustomerRef
```

Note: RENAME and MODIFY are proprietary extensions to SQL-92.

Compatibility Information

The LANGUAGE clause is not supported by TurboDB Managed.

See also

Column Data Types

1.6.4.5.3 CREATE INDEX Statement

Creates a new index for an existing table.

Syntax

```
CREATE [UNIQUE] INDEX index_reference ON table_reference
(column_reference [ASC|DESC] [,column_reference [ASC|DESC] ...] )
```

Description

Indexes are used to speed up query execution. Use the CREATE INDEX command to add a new index to an existing table:

```
CREATE INDEX OrderIdIdx ON Orders (OrderId ASC)
```

You may create multi-level hierarchical indexes as well:

```
CREATE INDEX TargetDateIdx ON Orders (DeliveryDate DESC, OrderId)
```

Use `UNIQUE` to create an index that raises an error if rows with duplicate column values are inserted. By default, indexes are not unique.

Note

The ASC and DESC attribute at column level is a proprietary extension to SQL-92.

1.6.4.5.4 CREATE FULLTEXTINDEX Statement

Creates a new full-text index for an existing table.

Syntax

```
CREATE FULLTEXTINDEX index_reference ON table_reference
(column_reference [, column_reference ...]) DICTIONARY table_reference
[CREATE] [UPDATE]
```

Description

A full-text index enables searching with full-text search-conditions like the CONTAINS predicate. The column references are a list of table columns of all types with the exception of blob columns including memos and wide memos. Full-text indexes need an additional database table, which contains the list of indexed words, the dictionary.

The dictionary table can be created by this statement or explicitly. If CREATE is not indicated, the statement expects an existing dictionary table with the following characteristics:

- First column is a *VARCHAR* or *VARWCHAR* of arbitrary length. This column determines the possible search-words. If words are longer than this column allows, they are cut.

- Second column is of type *BYTE*. It contains the global relevance of this word.
- Other columns may or may not follow according to the needs of the application.
- There must be a column of type *AUTOINC* to identify the words. The indication of this *AUTOINC* column must be the first column of the table.

If the CREATE clause is included, the statement creates a new dictionary table with a first column as *VARCHAR(20)*.

If UPDATE is included, words that are not found in the dictionary table are added to it. If UPDATE is not included, only words from the dictionary table are indexed and can be found in searches.

Note

Full-text search technology for tables up to level three is different from the one for tables starting from level four. Only full-text indexes for tables of level four and above support automatic maintenance and relevance calculation.

1.6.4.5.5 DROP Statement

Delete a table or an index from the database.

Syntax

```
DROP TABLE table_reference
```

```
DROP INDEX table_reference.index_name
```

```
DROP FULLTEXTINDEX table_reference.index_name
```

Description

Use DROP to completely remove the database object and all appropriate files from the database.

```
DROP INDEX Orders.OrderIdIdx
```

```
DROP TABLE Orders
```

Warning

While an index deleted by error can be re-created easily, the data in a dropped table is gone and cannot be restored.

1.6.4.5.6 UPDATE INDEX/FULLTEXTINDEX Statement

Repairs an index or full-text index.

Syntax

```
UPDATE INDEX table_reference.index_name
```

```
UPDATE FULLTEXTINDEX table_reference.index_name
```

Description

If an index seems out-of-date (this can happen in the embedded version of TurboDB, if the client application crashes), the index can be re-built using this command. Use the asterisk * for the *index_reference* to update all indexes of a table.

Note

The UPDATE FULLTEXTINDEX statement is available only for the new maintained full-text indexes for level 4 tables.

Example

```
UPDATE INDEX MyTable.*; UPDATE FULLTEXTINDEX MyTable.*
```

1.6.4.5.7 TurboSQL Column Types

These are the column types supported by *TurboSQL*.

AUTOINC

Syntax

```
AUTOINC(indication)
```

TurboDB Column Type

AutoInc

Description

Integer field which receives a unique number from the database engine. Field values of link columns in dependent tables are displayed according to the indication, which is a string containing an index definition. (See "Automatic Linking".)

Example

```
AUTOINC('LastName, FirstName')
```

BIGINT

Syntax

```
BIGINT [NOT NULL]
```

TurboDB Column Type

BigInt

Description

An integral number between –2^63 and +2^63–1

Example

```
BIGINT DEFAULT 4000000000
```

BIT

Not supported in TurboSQL.

BOOLEAN

Syntax

```
BOOLEAN [NOT NULL]
```

TurboDB Column Type

Boolean

Description

Possible values are TRUE and FALSE

Example

```
BOOLEAN DEFAULT TRUE
```

BYTE

Syntax

```
BYTE [NOT NULL]
```

TurboDB Column Type

Byte

Description

An integral number between 0 and 255

Example

```
BYTE NOT NULL DEFAULT 18
```

CHAR

Syntax

```
CHAR(n) [NOT NULL]
```

TurboDB Column Type

String

Description

Ansi string up to N characters long. 1 <= n <= 32767

Example

```
CHAR(40)
```

CURRENCY

Not supported in TurboSQL, use DOUBLE PRECISION or BIGINT.

DATE

Syntax

```
DATE [NOT NULL]
```

TurboDB Column Type

Date

Description

Date value between 1/1/1 and 12/31/9999

Example

```
DATE
```

DECIMAL

Not supported in TurboSQL, use DOUBLE PRECISION.

DOUBLE PRECISION

Syntax

```
DOUBLE [PRECISION][(p)] [NOT NULL]
```

TurboDB Column Type

Float

Description

Floating point number from 5.0e-324 to 1.7 x 10e308 with a precision of 12 signification digits. p is the number of displayed digits after the decimal point. 0 <= p <= 12.

Example

```
DOUBLE(4) NOT NULL
```

ENUM

Syntax

```
ENUM(value1, value2, value3, ...) [NOT NULL]
```

TurboDB Column Type

Enum

Description

Column that holds one of the enumeration values given stored as a number internally. The values must be valid identifiers up to 40 characters in length. There can be up to 15 values.

Example

```
ENUM(Red, Blue, Green, Yellow)
```

FLOAT

Not supported in TurboSQL, use DOUBLE PRECISION.

GUID

Syntax

```
GUID [NOT NULL]
```

TurboDB Column Type

Guid

Description

A universally unique identifier 16 bytes in size.

Example

```
GUID DEFAULT '12345678-abcd-abcd-efef-010101010101'
```

INTEGER

Syntax

```
INTEGER [NOT NULL]
```

TurboDB Column Type

Integer

Description

An integral number between –2.147.483.648 and +2.147.483.647

Example

```
INTEGER NOT NULL
```

LINK

Syntax

```
LINK(table_reference) [NOT NULL]
```

TurboDB Column Type

Link

Description

Holds value of *AutoInc* column of another table and such builds a one-to-many relationship. *Table_reference* is the name of referenced table. (See "Automatic Linking".)

Example

```
LINK(PARENTTABLE)
```

LONGVARBINARY

Syntax

```
LONGVARBINARY [NOT NULL]
```

TurboDB Column Type

Blob

Description

Long bit-stream containing arbitrary data up 2 GB

Example

```
LONGVARBINARY
```

LONGVARCHAR

Syntax

```
LONGVARCHAR [NOT NULL]
```

TurboDB Column Type

Memo

Description

Long Ansi string of variable length up to 2 G characters

Example

```
LONGVARCHAR
```

LONGVARWCHAR

Syntax

```
LONGVARWCHAR [NOT NULL]
```

TurboDB Column Type

WideMemo

Description

Long Unicode string of variable length up to 2 G characters

Example

```
LONGVARWCHAR
```

MONEY

Not supported in TurboSQL, use DOUBLE PRECISION or BIGINT.

NUMERIC

Not supported in TurboSQL, use DOUBLE PRECISION.

RELATION

Syntax

```
RELATION(table_reference)
```

TurboDB Column Type

Relation

Description

Holds any number of AutoInc values of another table. Used to create a many-to-many relationship. (See "Automatic Linking".)

Compatibility

Feature is not supported in TurboDB Managed 1.x.

Example

```
RELATION(PARENTTABLE)
```

TIME

Syntax

```
TIME[(p)] [NOT NULL]
```

TurboDB Column Type

Time

Description

Time of day with a precision of p, where p = 2 means minutes, p = 3 means seconds and p = 4

means milliseconds, the default being 3. The precision is available only in table level 4 and above; it is always set to 2 in tables up to level 3.

Example

```
TIME(4) DEFAULT 8:32:12.002
```

TIMESTAMP

Syntax

```
TIMESTAMP [NOT NULL]
```

TurboDB Column Type

DateTime

Description

Combined date and time with a precision of milliseconds between 1/1/1 12:00:00.000 am and 12/31/9999 11:59:59.999 pm

Examples

```
TIMESTAMP DEFAULT 23.12.1899_15:00:00
TIMESTAMP DEFAULT '5/15/2006_7:00:00'
```

VARCHAR

Syntax

```
VARCHAR(n) [NOT NULL]
```

TurboDB Column Type

String

Description

Same as CHAR

Example

```
VARCHAR(40)
```

VARWCHAR

Syntax

```
VARWCHAR(n) [NOT NULL]
```

TurboDB Column Type

WideString

Description

Same as WCHAR

Example

```
VARWCHAR(20) NOT NULL
```

SMALLINT

Syntax

```
SMALLINT [NOT NULL]
```

TurboDB Column Type

SmallInt

Description

An integral number between -32.768 and +32.767

Example

```
SMALLINT
```

WCHAR

Syntax

```
WCHAR(n) [NOT NULL]
```

TurboDB Column Type

WideString

Description

Unicode string up N characters long. The actual field size in bytes is twice the number of characters. 1 <= n <= 32767

Example

```
WCHAR(1000) DEFAULT '-'
```

1.6.4.6 Programming Language

This feature is only available in TurboDB Managed.

TurboSQL provides language elements to create routines that can be called from SQL commands.

User-defined functions

Functions are used to simplify SQL commands and to provide additional functionality. They can be coded either in TurboSQL or be imported from a .NET assembly. Functions can only have input parameters and always calculate a return type. They must not have side-effects. Functions are managed using the CREATE FUNCTION and DROP FUNCTION statements. Functions can be used for computed indexes, computed columns and checks.

User-defined procedures

Procedures are used to call complex sequences of SQL statements with a single statement. For example, using a procedure, multiple rows can be updated in different tables in one step. Procedures can be implemented either in TurboSQL or be imported from a .NET assembly. Procedures are managed using the CREATE PROCEDURE and DROP PROCEDURE statements.

User-defined aggregates

Aggregates compute accumulated values in result set groups. They can be used for example to compute the 2nd order maximum, the standard deviation or other accumulated values from a grouped result set. Aggregates are implemented in a .NET assembly and managed with the CREATE AGGREGATE and DROP AGGREGATE statements.

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

Other topics

Exchanging parameters with .NET Assemblies

1.6.4.6.1 CALL Statement

Syntax

```
CALL procedure_name([argument, ...])
```

Description

Executes a stored procedure with the given arguments.

Sample

```
CALL LogLine('This is a test statement')
```

Compatibility Information

This statement is only available in TurboDB Managed.

1.6.4.6.2 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
```

Description

A function can be called wherever a scalar value is expected, in select elements, in search conditions and all other kinds of expressions.

namespace_name.class_name refers to a public class in the assembly.

method_name refers to the name of a static public function in the assembly. The function must not be overloaded and the parameters must fit the parameters of the TurboSQL function (see "Exchanging Parameters with .NET Assemblies").

Samples

```
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
```

Compatibility Information

This statement is only available in TurboDB Managed.

1.6.4.6.3 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
```

Description

namespace_name.class_name refers to a public class in the assembly.

method_name refers to the name of a static public function in the assembly. The function must not be overloaded and the parameters must fit the parameters of the TurboSQL function (see "Exchanging Parameters with .NET Assemblies").

Samples

```
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
```

The following example requires a static public method *Send* of a public class *SmtpClient* in the namespace *Email* in an assembly called *Email.dll*. The assembly must be located in the same directory as is the database (tdbd) file.

```
CREATE PROCEDURE SendEmail(:Address WCHAR(100), :Subject WCHAR(100), :
Content WCHAR(100)) AS
EXTERNAL NAME [Email.SmtpClient].Send,Email
```

Compatibility Information

This statement is only available in TurboDB Managed.

1.6.4.6.4 CREATE AGGREGATE Statement

Syntax

```
CREATE AGGREAGATE aggregate_name([:parameter_name data_type]...) RETURNS
data_type AS EXTERNAL NAME [namespace_name.class_name],assembly_name
```

Description

namespace_name.class_name refers to a public class in the assembly. This class must have a default constructor and implement three methods:

```
[C#]
public <class_name>()
public void Init()
public void Accumulate(<data_type>)
public <data_type> Terminate()
```

The *data_type* must fit the parameters of the TurboSQL function (see "Exchanging Parameters with .NET Assemblies").

When a user-defined aggregate is used in a SQL statement, e.g. *SELECT MAX2(Salary) FROM Employees*, the execution engine creates an instance of the CLR aggregation class. At the beginning of each group it calls the *Init* function. After this, it calls the *Accumulate* function for each row in the group in the order defined by the GROUP BY clause. After all rows of the group have been accumulated, the execution engine calls the *Terminate* function to retrieve the result. Any resources allocated can be disposed of in the *Terminate* function.

Samples

The C# code for an aggregate that computes the second order maximum looks like this.

```
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;
    }
}
```

And this is how you import the aggregate into TurboSQL.

```
CREATE AGGREGATE Max2(:x DOUBLE) RETURNS DOUBLE AS
EXTERNAL NAME [MathRoutines.SecondOrderMax],MathRoutines
```

Compatibility Information

This statement is only available in TurboDB Managed.

1.6.4.6.5 DROP FUNCTION/PROCEDURE/AGGREGATE Statement

Syntax

```
DROP FUNCTION | PROCEDURE | AGGREGATE
```

Description

Use DROP to remove the function, procedure or aggregate from the database.

Compatibility Information

This statement is only available in TurboDB Managed.

1.6.4.6.6 DECLARE Statement

Syntax

```
DECLARE :variable_name data_type
```

Samples

```
DECLARE :i INTEGER
DECLARE :s CHAR(300)
```

Compatibility Information

This statement is only available in TurboDB Managed.

1.6.4.6.7 IF Statement

Syntax

```
IF search_condition THEN if_statement [ELSE else_statement]
```

Description

Executes the *if_statement* if and only if *search_condition* evaluates to True. If *search_condition* does not return True and *else_statement* is given, it is executed.

Sample

```
IF :s <> '' THEN SET :s = :s + ', '
```

```
IF :a > 0 THEN BEGIN
     SET :r = SQRT(:a)
END ELSE BEGIN
     SET :r = SQRT(-:a)
END
```

Compatibility Information

This statement is only available in TurboDB Managed.

1.6.4.6.8 SET Statement

Syntax

```
SET :variable_name = expression
```

Description

Assigns a new value to the variable.

Example

```
SET :LastName = 'Miller'
```

Compatibility Information

The SET statement is only available in TurboDB Managed.

1.6.4.6.9 WHILE Statement

Syntax

```
WHILE search_condition statement
```

Description

Executes statement as long as *search_condition* evaluates to True.

Sample

```
DECLARE :I INTEGER
WHILE :I < 100 SET :I = :I + 1
```

```
DECLARE :I INTEGER
WHILE :I < 100 BEGIN
     SET :I = :I + 1
END
```

Compatibility Information

This statement is only available in TurboDB Managed.

1.6.4.6.10 Exchanging Parameters with .NET Assemblies

Assemblies that are used for external functions, procedures or aggregates must be located in the same directory as the database file.

When calling methods from .NET assemblies, parameters are mapped from TurboSQL to CLR according to the table below. Regarding this table, there is a difference between functions on one hand and procedures and aggregates on the other hand.

Functions have only input parameters and are evaluated to NULL, if one of its arguments is NULL. In this case, the CLR method is not executed at all. Therefore, NULL will never be passed to user-defined CLR functions and the CLR method must be declared with the non-nullable CLR types only. For example,

```
CREATE FUNCTION Log2(:x DOUBLE NOT NULL) RETURNS DOUBLE NOT NULL AS
EXTERNAL NAME [MyNamespace.MyClass].MyMethod,MyAssembly
```

corresponds to this definition in C#:

```
public class MyClass {
     static public double MyMethod(double x) {...}
}
```

Because the argument is not nullable, *Log2* can never be used on nullable arguments. However, if declared like this

```
CREATE FUNCTION Log2(:x DOUBLE) RETURNS DOUBLE AS EXTERNAL NAME
[MyNamespace.MyClass].MyMethod,MyAssembly
```

the same definition as above is valid in C#. When *Log2* is called with a NULL argument, the execution engine will return NULL without calling the CLR method.

This rule only holds for functions; CLR procedures and CLR aggregates are always called, even if one of the arguments is NULL. Therefore, the parameters and the return type of the CLR definition must be able to transport the NULL value. TurboSQL does this by passing a *null* value (*Nothing* in Visual Basic), which is obvious for CHAR types and array types like *LONGVARBINARY*. In order to be able to do this with value types like *Int64* or *DateTime*, the nullable wrapper must be used.

```
CREATE PROCEDURE TestProc(:x DOUBLE) AS EXTERNAL NAME [MyNamespace.
MyClass].MyMethod,MyAssembly
```

is therefore mapped by

```
public class MyClass {
     static public void MyMethod(Nullable<double> x) {...}
}
```

or

```
public class MyClass {
     static public void MyMethod(double? x) {...}
}
```

or in Visual Basic:

```
Public Class MyClass
     Public Shared Sub MyMethod(X As Nullable(Of Double))
          ...
     End Sub
End Class
```

For more information on nullable wrappers, search MSDN for the *Nullable* class.

TurboSQL type	Non-Nullable CLR type	Nullable CLR type
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>*

Compatibility Information

This statement is only available in TurboDB Managed.

1.6.5 TurboDB Products and Tools

For developers using TurboDB there is a set of additional tools:

TurboDB Viewer Visual tool for managing TurboDB database tables and indexes, editing database tables. Included in TurboDB packages.

TurboDB Pilot SQL console for managing and working with level 5 single-file databases.

dataweb Compound File Explorer: Visual tool for managing the storage objects within a dataweb compound file like e.g. a TurboDB single-file database.

TurboDB Workbench: Console application for creating, indexing, altering TurboDB tables. Included in TurboDB Components package.

TurboDB Data Exchange: Importing and exporting records to and from TurboDB tables.

TurboDB Server: Database server for TurboDB. Allows up to 100 concurrent sessions on the same database.

TurboDB Studio: Complete RAD tool for developing Windows applications with TurboDB Engine.

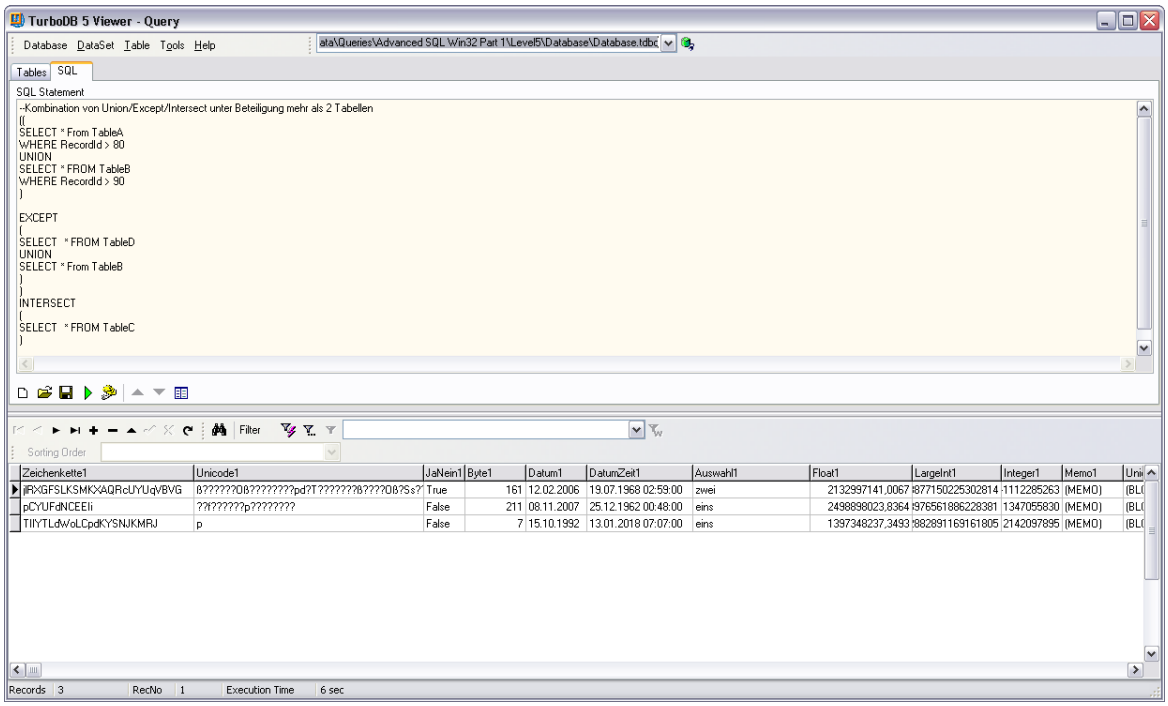
More information on TurboDB products and tools is to be found on the dataweb homepage www.dataweb.de.

1.6.5.1 TurboDB Viewer

TurboDB Viewer is a visual tool for managing tables and indexes and for editing tables. Here is what you can do using TurboDB Viewer:

- View and edit tables.
- Execute SQL queries.
- Create, alter and maintain database tables.
- View and define checks and foreign keys.
- Create and remove indexes.
- Create and remove full-text indexes.

TurboDB Viewer is included in all TurboDB Editions. Since it is itself written using the native version of TurboDB, it supports all table levels and features.

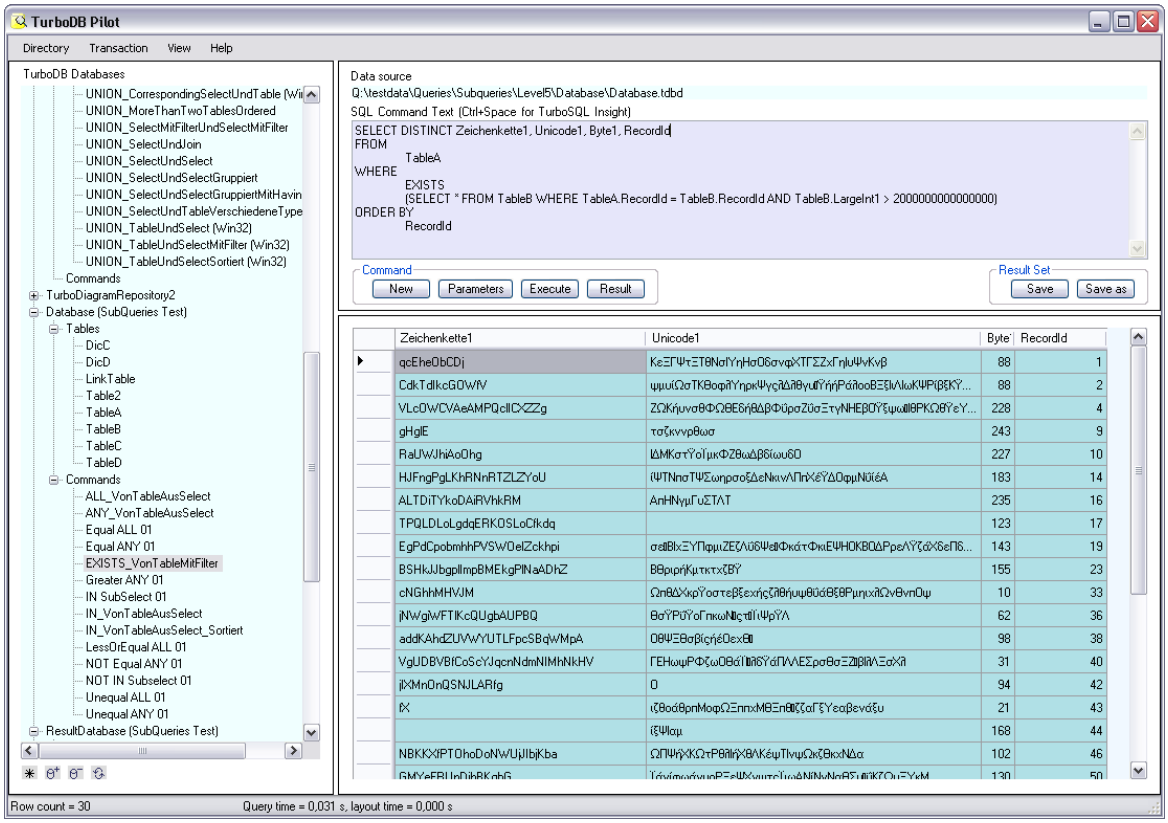


When using TurboDB Viewer to create databases for TurboDB Managed, you must however be careful to not use features, that are unsupported by TurboDB Managed. For TurboDB Managed you may want to use TurboDB Pilot, which is based on the managed database engine and is strictly SQL-based.

1.6.5.2 TurboDB Pilot

TurboDB Pilot is an SQL console for TurboDB level 5 databases written with the managed database engine. It is a complete tool for managing TurboDB level 5 databases.

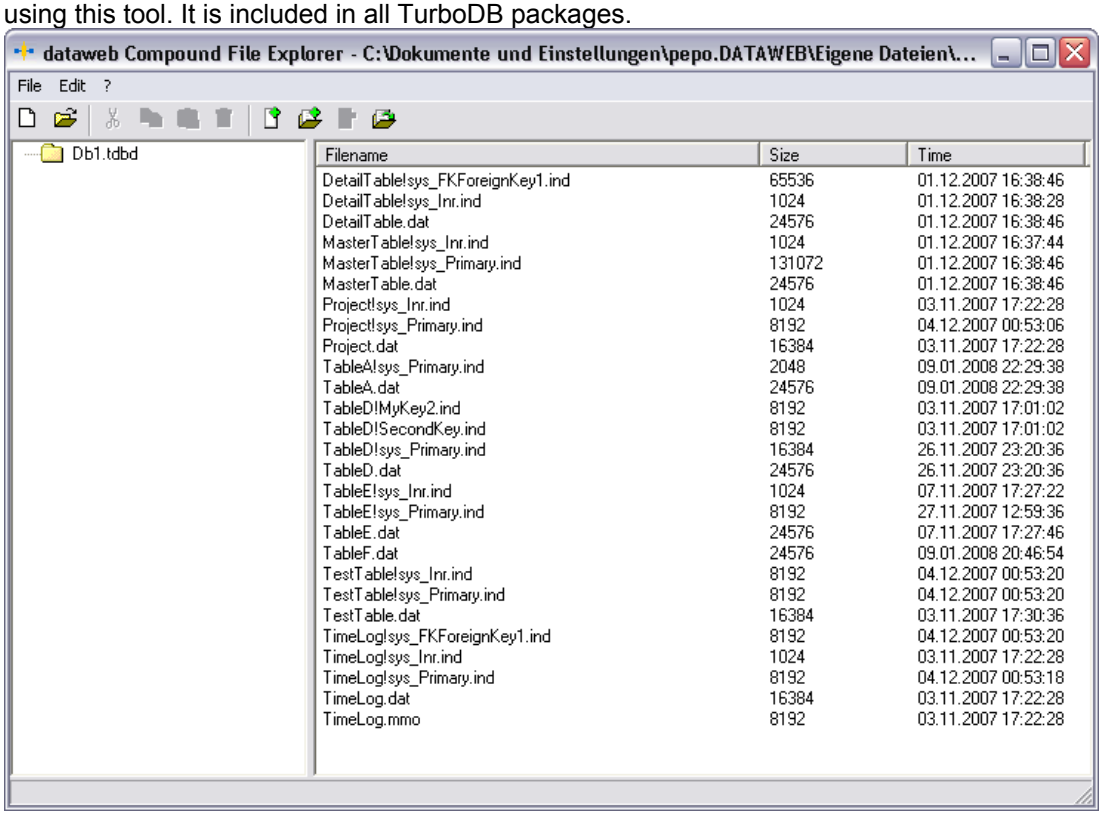
- Creating and altering tables
- Performing queries with parameters
- Updating databases
- Examining schema information



TurboDB Pilot stores a directory of all databases in your system and therefore makes it easy to retrieve them.

1.6.5.3 dataweb Compound File Explorer

This is the managing tool for dataweb's compound file technology. Since a TurboDB single-file database is a compound file, you can view and edit the storage objects of a single-file database using this tool. It is included in all TurboDB packages.



1.6.5.4 TurboDB Workbench

tdbwkb is a small text-based free-ware tool for managing TurboDB tables. It is available for Windows and Linux and can be downloaded at http://www.turbodb.de/. *tdbwkb* offers commands for

- Creating new tables
- Modifiying existing tables
- Show the table structure of existing tables
- Creating indexes for a table
- Deleting indexes for a table
- Repair a table and its indexes
- Deleting a table
- Switching between different databases

Running *tdbwkb* will show the copyright and the *tdbwkb* prompt where you can enter the different commands. Enter *help* to show a list of available commands.

Here is a sample *tdbwkb* session to illustrate the available features.

```
home/usr1>tdbwkb
dataweb Turbo Database Workbench Version 4.0.1 (TDB 6.1.6)
Copyright (c) 2002-2003 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
```

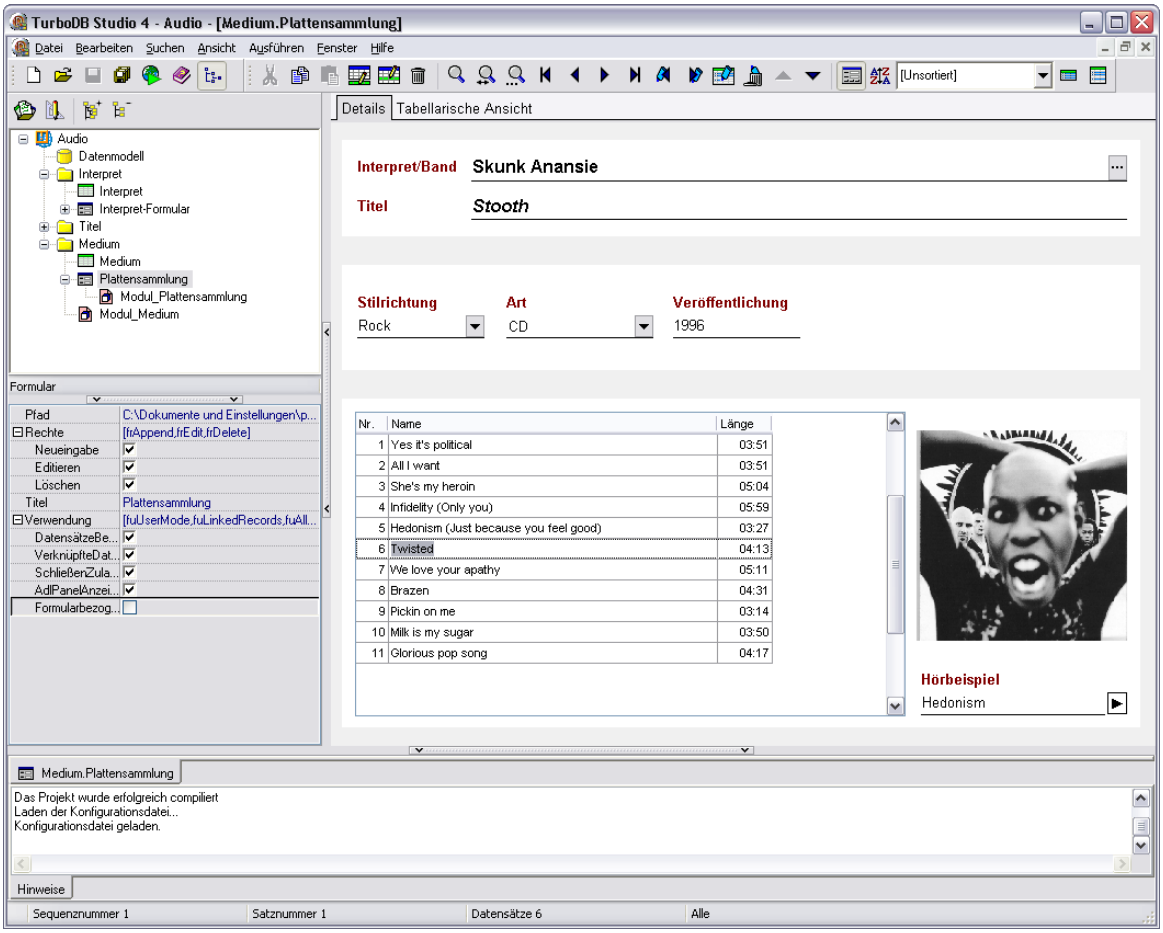
```
  5 M    Description
  6 N    RecordId
Indexes:
animals.inr                    RecordId:4
animals.id                     Name:40
byfamily.ind                   Family:40, Name:40
```

1.6.5.5 TurboDB Studio

This is our tool for creating Windows client applications for Turbo Database. Using TurboDB Studio you create forms and reports for interactive applications and printing. TurboDB Studio enables you to create customized executables with your own name, splash screen, menus and toolbars in a couple of hours. Users of TurboDB can manage their TurboDB database very quickly, enter or modify data and print all kind of reports.

You can use TurboDB Studio to

- Manage TurboDB database more comfortably then with TurboDB Viewer
- Prototype your TurboDB applications very rapidly
- Provide additional customized management tools to the users of TurboDB application (includes reporting)
- Create full-sized Windows applications based on TurboDB



More information on *TurboDB Studio* can be found on the the dataweb homepage (currently only in German).

1.6.5.6 TurboDB Data Exchange

Another text-based free-ware tool that reads and writes data from and to TurboDB tables. Formats supported by *tdbDataX* are Text, dBase, TurboDB, XML and ADO. You can download it from http://www.turbodb.de/.

1.7 Developing with TurboDB

This topic explains concepts of the TurboDB database engine.

1.7.1 Database Connections

As described in "Single-File Databases and Database Directories", TurboDB supports two types of databases. The more traditional one, where each table and index has its own file on the hard disk and one, where all data is contained in one single file. In order to connect to a database, you have to set the DataSource property of the TurboDBConnection to either the database directory or to the database file. E.g:

```
// This opens a directory database
TurboDBConnection connection = new
TurboDBConnection("c:\mydatabasedirectory\");
connection.Open();
// Now you can access database tables in *.dat files in this directory
```

```
// This opens a database file
TurboDBConnection connection = new
TurboDBConnection("c:\mydatabases\database1.tdbd");
connection.Open();
// Now you can access database tables within the database file
```

In order to create a database file, you can use the method *CreateDatabase* of *TurboDBConnection* .

A TurboDB database connection can be opened read-only and/or exclusive. If you open the database connection in read-only mode, UPDATE, INSERT and DELETE commands will fail. If you open the database connection in exclusive mode, other applications and other threads will not be able to open the same database.

If you want to open a database connection for a database, which is located on a read-only medium (e.g. on a CD) then you must set the *ReadOnly* and the *Exclusive* properties to true.

See also

Single-File Databases and Database Directories

1.7.2 SQL Dialect

The SQL dialect used with TurboDB is TurboSQL, a subset of ANSI SQL 92 and a superset of the SQL dialect for ODBC. A full reference can be found in the "TurboSQL Guide".

1.7.3 Protected Database Tables

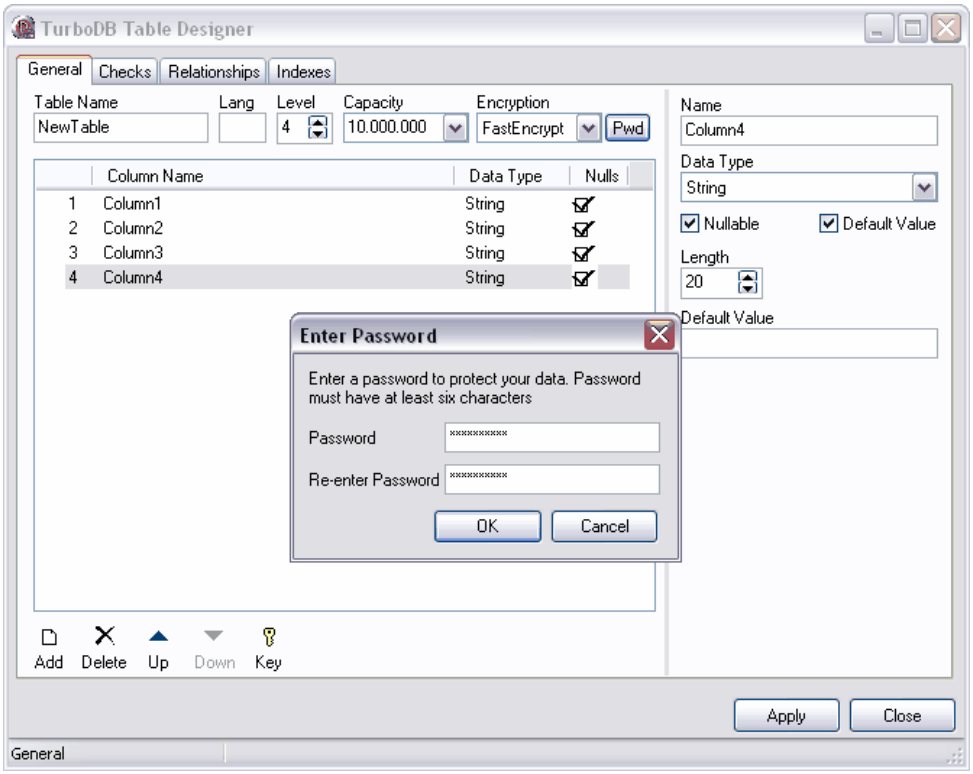
Feature supported only in TurboDB Win32 for ADO.NET

TurboDB offers multiple ways of prohibiting unauthorized access to your database table. An overview can be found in the section on data security.

How to Create a Protected Table with TurboDB Viewer

When creating a new table with TurboDB Viewer, you select the encryption method click on the button *Pwd* to enter the password. If you selected the classic mode, the alphanumeric password

and the numeric key must be entered in one text box separated by semicolon.



How to Create a Protected Table in Code

If you create the database tables from within your application, you can add the ENCRYPTION and the PASSWORD clauses to the CREATE TABLE (or ALTER TABLE) statement, as described in "CREATE TABLE Command".

Using a Protected Table

When you are accessing a protected table in your application, you must find a way to provide the password and key, when a table is opened. This is done using the *PasswordNeeded* event of the *TurboDBConnection* component.

1.7.4 Using TurboDB with ASP.NET

You can use TurboDB in your ASP.NET applications exactly like you do it with other databases. It is a good idea to create a subfolder (e.g. called db) in the application directory under *wwwroot* and put the database file(s) there.

Depending on the authorization scheme of your application you must give file access rights (probably full access) to this database directory for the *ASPNET* user or for everyone. If you miss to do so, you will receive an exception saying that TurboDB cannot open the database.

1.8 ADO.NET Data Provider Reference

The ADO.NET providers for TurboDB for .NET and for TurboDB Managed are very similar and therefore documented together in this chapter. The main difference is the namespace.

DataWeb.TurboDB Namespace for TurboDB for .NET

DataWeb.TurboDBManaged Namespace for TurboDB Managed

1.8.1 TurboDB Namespace

The namespace *DataWeb.TurboDB* contains the TurboDB ADO.NET Data Provider for TurboDB for .NET.

The namespace *DataWeb.TurboDBManaged* contains the TurboDB ADO.NET Data Provider for TurboDB Managed.

The types in those namespaces are the same, but some of them do not exist in both namespaces.

Standard ADO.NET Provider Classes

TurboDBConnection	Represents a connection to a TurboDB database.
TurboDBDataReader	Offers a way to read forward through a result set.
TurboDBCommand	Encapsulates a TurboSQL command.
TurboDBDataAdapter	Contains a set of commands to fill and update a dataset
TurboDBParameterCollection	Holds the list of SQL parameters for a command.
TurboDBParameter	Stores the properties of a single SQL command.
TurboDBException	Describes an error condition with TurboDB.
TurboDBCommandBuilder	Creates UPDATE, INSERT and DELETE commands from a SELECT command.

Enumerations

TurboDBType	Indicates the data type of a TurboDB database columns.
TurboDBLockType	Defines the type of locking applied to a TurboDB table. TurboDB Native only

1.8.2 TurboDBConnection Class

Represents an open connection to a TurboDB database.

A list of all members of this type can be found in *TurboDBConnection* Members.

C#

```
public sealed class TurboDBConnection: DbConnection, ICloneable
```

Visual Basic

```
Public NotInheritable Class SqlConnection Inherits DbConnection Implements ICloneab
```

See also

TurboDBConnection Members

1.8.3 TurboDBConnection Members

TurboDB Connection - Overview

This topic documents the members, which are different from the .NET framework classes. Refer to the Microsoft documentation for all members not explicitly listed here.

Public Properties

DataSource	Name of a TurboDB database file or a database directory
Exclusive	Defines whether other applications can open the same database.

ReadOnly	Indicates non-writing access to database.

Public Methods

CreateDatabase	Creates a TurboDB database file

Public Events

PasswordNeeded	Occurs when a secured or encrypted table should be opened.

See also

TurboDBConnection class

1.8.4 TurboDBConnection.ConnectionString

Indicates a textual representation of the parameters used to open a database connection.

C#

```
public override string ConnectionString {get; set;}
```

Visual Basic

```
Public Overrides Property ConnectionString As String
```

Description

The connection string can have three fields:

Name	Description	Typical Value
DataSource	File name of the database file or directory name of the database directory (see "Databases")	C:\TurboDB\MyDb1.tdbd
Exclusive	Defines, whether other applications can access the same database at the same time.	false
ReadOnly	Defines, whether the application can execute modification statements (INSERT, UPDATE, DELETE) over this connection.	false

A typical connection string looks like this:
DataSource="C:\TurboDB\MyDb1.tdbd";Exclusive=True;ReadOnly=False

The values for *Exclusive* and *ReadOnly* can be set by properties as well. The *DataSource* is modified by the *ChangeDatabase* function.

Compatibility

TurboDB Managed only supports database files, the DataSource field cannot be a directory.

In TurboDB Managed, the Exclusive field has no effect, because TurboDB Managed always opens the database in exclusive mode.

See also

TurboDBConnection class | TurboDBConnection members

1.8.5 TurboDBConnection.CreateDatabase

Creates a new TurboDB database.

C#

```
public void CreateDatabase(TurboDBDatabaseType dbType)
```

Visual Basic

```
Public Sub CreateDatabase(dbType As TurboDBDatabaseType)
```

Description

dbType determines, whether a single-file or directory database is created. You must set the *DataSource* property to the file name for the new TurboDB database file or directory before calling *CreateDatabase*.

Compatibility

In version 4 and below this method did not have the parameter. To have the same effect, put *TurboDBDatabaseType.SingleFile* here. The method was expanded to make it possible to create directory-based databases as well.

In TurboDB Managed the argument to *dbType* must always be *TurboDBDatabaseType.SingleFile* because directory-based databases are not supported.

Sample

```
// This code creates a new TurboDB database file and opens it
TurboDBConnection turboDBConnection = new TurboDBConnection();
turboDBConnection.ConnectionString = @"DataSource=""c:\mydatabases\database98.tdbd"
turboDBConnection.CreateDatabase(TurboDBDatabaseType.SingleFile);
turboDBConnection.Open();
```

See also

TurboDBConnection class | TurboDBConnection members

1.8.6 TurboDBConnection.CompressDatabase

Compacts a TurboDB database file.

C#

```
public void CompressDatabase()
```

Visual Basic

```
Public Sub CompressDatabase
```

Description

CompressDatabase eliminates all free storage space from shortened or deleted storage objects within the database file. For performance reasons, this storage space is kept within the file and re-used for the next allocation request during normal operation. Sometimes, for example before a database is going to be deployed, it is useful to compact it in order to reduce its size on the disk.

See also

TurboDBConnection class | *TurboDBConnection* members

1.8.7 TurboDBConnection.DataSource

Indicates the TurboDB data source to connect with.

C#

```
public string DataSource {get;}
```

Notes

As of .NET Framework 2.0, DataSource is a read-only property because of its inheritance. In order to connect to a database you must set the *ConnectionString* property. The *DataSource* property will then indicate the database file or the database directory.

See also

TurboDBConnection class | TurboDBConnection members | Database connections | Single-file databases and database directories

1.8.8 TurboDBConnection.Exclusive

Defines whether other applications can open the same database.

C#

```
public bool Exclusive {set; get;}
```

Notes

You should set the property to true before you open the connection, if you want to access a data source exclusively. A database connection opened in exclusive mode will prevent other applications to access the same database. Besides, execution of commands will be faster. Exclusive access is necessary for databases on read-only directories e.g. for CD-based databases.

Compatibility

In TurboDB Managed this property has no effect, because the database is always opened exclusively.

See also

TurboDBConnection class | TurboDBConnection members

1.8.9 TurboDBConnection.PasswordNeeded Event

Occurs when a secured or encrypted table should be opened.

C#

```
public event TurboDBPasswordNeededEventHandler PasswordNeeded
```

Event Data

The event handler receives an argument of type TurboDBPasswordNeededEventArgs with these fields:

Property	Description
TableName	Name of the TurboDB table to open
Password	The password passed from the event handler to the connection
Retry	If true, the connection will try to open the table with the given password and key. Otherwise the connection will cancel trying to open the TurboDB table.

Notes

This event is issued each time, when the database engine tries to open a protected database table. This is normally the case during the execution of a SQL command. When handling the event, fill the *Password* field with the password for the table and set Retry to true.

Compatibility

In earlier versions of TurboDB there was an encryption mode, which required an alphanumeric password and a numeric key. This mode is called the classic mode and requires now to enter the former password and key within one password like this <password>;<key>, e.g: *secret;-123*.

TurboDB Managed does never call this event, because it currently has no support for encrypted or protected tables.

Sample

This is a possible implementation of a handler for the *PasswordNeeded* event, if your application knows the password and the code of all tables it is using. We assume here, that *Table1* had been encrypted with AES and *Table2* in classic mode.

C#

```
private void turboDBConnection1_PasswordNeeded(object sender, DataWeb.TurboDB.Turbo
{
  if (e.TableName.CompareTo("Table1") == 0) {
    e.Password = "Table1PW";
    e.Retry = true;
  } else if (e.TableName.CompareTo("Table2") == 0) {
    e.Password = "Table2PW;99999992";
    e.Retry = true;
  } else {
    MessageBox.Show("Don't know password and key for database table.");
    e.Retry = false;
  }
}
```

See also

TurboDBConnection Class | TurboDBConnection Members | Protected Database Tables | Data Security

1.8.10 TurboDBConnection.ReadOnly

Indicates non-writing access to the database.

C#

```
public bool ReadOnly {get; set;}
```

Notes

Set *ReadOnly* to true before opening a database connection, if you want to use the database in read-only mode only. This is necessary as well, if you are opening a database on a CD or on any other read-only medium.

See also

TurboDBConnection class | TurboDBConnection members

1.8.11 TurboDBDatabaseType

Indicates the type of a TurboDB database.

C#

```
[Serializable]
public enum TurboDBDatabaseType
```

Members

Name	Description
Directory	The database objects (tables, indexes, etc.) are separate files in a operating system directory.
SingleFile	The database contains all object within one single operating system file.

Requirements for TurboDB Native

Namespace: *DataWeb.TurboDB*
Platforms: Windows 98, Windows ME, Windows NT 4, Windows 2000, Windows XP
Version Information: .NET Framework 1.1, 2.0
Assembly: DataWeb.TurboDB (for .NET Framework 1.1) or DataWeb.TurboDB.Native20.Provider (for .NET Framework 2.0)

Requirements for TurboDB Managed on .NET Framework

Namespace: *DataWeb.TurboDBManaged*
Platforms: Windows 98, Windows ME, Windows NT 4, Windows 2000, Windows XP

Version Information: .NET Framework 2.0
Assembly: DataWeb.TurboDB.Managed20.Provider

Requirements for TurboDB Managed on .NET Compact Framework

Namespace: *DataWeb.TurboDBManaged*
Platforms: Windows CE, Windows Mobile for Pocket PC, Windows Mobile for Smartphone
Version Information: .NET Compact Framework 2.0
Assembly: DataWeb.TurboDB.CF20.Provider

1.8.12 TurboDBTransaction Class

Encapsulates the ability to begin and end transactions.

A list of all members of this type can be found in TurboDBTransaction Members.

C#

```
public sealed class TurboDBTransaction: DbTransaction
```

Notes

TurboDB supports only one isolation level for its transactions: read committed. When an application is terminated (e.g. crashes) during a transaction all changes are rolled-back automatically, when the tables are used the next time. This is called automatic crash recovery.

See also

TurboDBTransaction members | Transactions

1.8.13 TurboDBTransaction Members

TurboDBTransaction - Overview

This topic documents the members, which are different from the .NET framework classes. Refer to the Microsoft documentation for all members not explicitly listed here.

Public Properties

Connection	Gets the TurboDBConnection object associated with the transaction, or a null reference (Nothing in Visual Basic) if the transaction is no longer valid.
IsolationLevel	Specifies the IsolationLevel for this transaction. This value is always set to IsolationLevel.ReadCommited.

Public Methods

Rollback	Rolls back a transaction from a pending state.
Commit	Commits the database transaction.

See also

TurboDBTransaction class | TurboDBTransaction members

1.8.14 TurboDBTransaction.Commit

Commits the database transaction.

C#

```
public void Commit()
```

Exceptions

TurboDBException

Compatibility

The meaning of this method has changed completely between v4 and v5. See "Transactions".

See also

TurboDBTransaction class | TurboDB Transaction members

1.8.15 TurboDBTransaction.Rollback

Rolls back a transaction from a pending state.

C#

```
public void Rollback()
```

Exceptions

TurboDBException

Compatibility

The meaning of this method has been changed completely between v4 and v5. See "Transactions".

See also

TurboDBTransaction class | TurboDBTransaction members | TurboDBLockType

1.8.16 TurboDBDataReader Class

Offers a way to access data from a table or a result set in a uni-directional manner.

A list of all members of this type can be found in *TurboDBDataReader* Members.

C#

```
public sealed class TurboDBDataReader: DbDataReader
```

See also

TurboDBDataReader Members

1.8.17 TurboDBDataReader Members

TurboDBDataReader - Overview

This topic documents the members, which are different from the .NET framework classes. Refer to the Microsoft documentation for all members not explicitly listed here.

Public Properties

Depth	Always 1.
RecordsAffected	Returns the number of records updated, inserted or deleted by the command.

Public Methods

Close	Closes the TurboDBDataReader
GetDataTypeName	Returns a string indicating the name of the TurboDB data type for a column.
GetSchemaTable	Returns a DataTable, which describes the column meta data of the TurboDBDataReader.

See also

TurboDBDataReader class

1.8.18 TurboDBDataReader.GetSchemaTable

Returns a *DataTable*, which describes the column meta data of the *TurboDBDataReader*.

C#

```
public DataTable GetSchemaTable()
```

Return Value

A *DataTable* describing the structure of the current result set of the data reader.

Implements

IDataReader.GetSchemaTable

Exceptions

InvalidOperationException The *TurboDBDataReader* is closed.

Hints

Each row in the schema table describes one column in the result set. The schema table has these columns:

Name	Data Type	Description
ColumnName	*string*	The name of the column in the result set
Column Ordinal	*int*	The ordinal number of the column in the result set
ColumnSize	*int*	Maximum size of the data in the column
NumericPrecision	*int*	Numeric precision, if column data type is a real number
Numeric Scale	*int*	Number of digits to the right of the decimal separator (for real numbers)
DataType	*Type*	The .NET framework type of the column
ProviderType	*TurboDBType*	The TurboDB type of the column
IsLong	*bool*	True, if column is Blob or Memo
AllowDBNull	*bool*	True, if no value (i.e. a Null value) is allowed for the column
IsReadOnly	*bool*	True, if column must not be written to
IsRowVersion	*bool*	True, if column is of type AutoInc
IsUnique	*bool*	True, if there is a unique index on this column alone
IsKey	*bool*	True, if there is a unique index, whose fields are all in the result set and this column is one of them
BaseSchemaName	*string*	Always empty
BaseCatalogName	*string*	Always empty
BaseTableName	*string*	Name of database table, where this column is taken from
BaseColumnName	*string*	Name of this column in BaseTableName

See also

TurboDBDataReader Class | *TurboDBDataReader* Members

1.8.19 TurboDBCommand Class

Holds a table name or TurboSQL command for executing against the database.

A list of all members of this type can be found in *TurboDBCommand* members.

C#

```
public sealed class TurboDBCommand: DbCommand
```

See also

TurboDBCommand members

1.8.20 TurboDBCommand Members

TurboDBCommand - Overview

This topic documents the members, which are different from the .NET framework classes. Refer to the Microsoft documentation for all members not explicitly listed here.

Public Properties

Parameters	Indicates the *TurboDBParameterCollection* for this command.
Transaction	Not yet implemented.

For other members please refer to the documentation of *IDbCommand*.

See also

TurboDBCommand class

1.8.21 TurboDBDataAdapter Class

Contains a set of TurboSQL commands, which are used to fill and update a data set.

A list of all members of this type can be found in TurboDBDataAdapter members.

C#

```
public sealed class TurboDBDataAdapter: DbDataAdapter, IDbDataAdapter
```

See also

TurboDBDataAdapter members

1.8.22 TurboDBDataAdapter Members

TurboDBAdapter - Overview

This topic documents the members, which are different from the .NET framework classes. Refer to the Microsoft documentation for all members not explicitly listed here.

Public Properties

InsertCommand	Indicates the TurboDBCommand used for inserts.
UpdateCommand	Indicates the TurboDBCommand used for updates.
DeleteCommand	Indicates the TurboDBCommand used for deletes.
SelectCommand	Indicates the TurboDBCommand used for selects.

See also

TurboDBAdapter class

1.8.23 TurboDBCommandBuilder Class

Creates appropriate UPDATE, INSERT and DELETE statements based on a SELECT statement.

A list of all members of this type can be found in TurboDBCommandBuilder Members.

C#

```
public sealed class TurboDBCommandBuilder: DbCommandBuilder
```

See also

TurboDBCommandBuilder Members

1.8.24 TurboDBCommandBuilder Members

TurboDBCommandBuilder - Overview

This topic documents the members, which are different from the .NET framework classes. Refer to the Microsoft documentation for all members not explicitly listed here.

Public Methods

DeriveParameters	Not yet implemented.

See also

TurboDBCommandBuilder class | DataWeb.TurboDB namespace

1.8.25 TurboDBParameterCollection Class

A list of parameter objects for a TurboSQL command and the relationships to the columns of a data set.

A list of all members of this type can be found in TurboDBParameterCollection Members.

C#

```
public sealed class TurboDBParameterCollection: MarshalByRefObject, IDataParameterC
```

Visual Basic

```
Public NotInheritable Class SqlParameterCollection Inherits DbParameterCollection
```

See also

TurboDBParameterCollection Members

1.8.26 TurboDBParameterCollection Members

TurboDBParameter - Overview

This topic documents the members, which are different from the .NET framework classes. Refer to the Microsoft documentation for all members not explicitly listed here.

See also

TurboDBParameterCollection | DataWeb.TurboDB namespace

1.8.27 TurboDBParameter

Represents a parameter in a TurboSQL statement and how it refers to a data set column.

A list of all members of this type can be found in TurboDBParameter Members.

C#

```
public sealed class TurboDBParameter: DbParameter, IDbDataParameter, IDataParameter
```

See also

TurboDBParameter Members

1.8.28 TurboDBParameter Members

TurboDBParameter - Overview

This topic documents the members, which are different from the .NET framework classes. Refer to the Microsoft documentation for all members not explicitly listed here.

See also

TurboDBParameter class | DataWeb.TurboDB namespace

1.8.29 TurboDBException

This exception is thrown when the TurboDB ADO.NET Data Provider issues a warning or an error.

A list of all members of this type can be found in TurboDBException members.

C#

```
public sealed class TurboDBException: ApplicationException
```

See also

TurboDBException members

1.8.30 TurboDBException Members

TurboDBException - Overview

This topic documents the members, which are different from the .NET framework classes. Refer to the Microsoft documentation for all members not explicitly listed here.

See also

TurboDBException class

1.8.31 TurboDBType Enumeration

Indicates the type of a field, a property or a parameter.

C#

```
[Serializable]
public enum TurboDBType
```

Members

Name	Native Type	Description
String	String	Ansi string up 255 characters
SmallInt	SmallInt	2 byte integral number
Float	Float	8 byte floating point number
Byte	Byte	1 byte positive integral number
Boolean	Boolean	True or false
Date	Date	Date
Memo	Memo	Ansi string of arbitrary length

Enum	Enum	One out of a list of available values
AutoInc	AutoInc	Automatically assigned unique integer
Link	Link	One-to-many relationship to another table
Blob	Blob	Binary data of arbitrary size
Relation	Relation	Many-to-many relationship to another table
Time	Time	Time of day in hours and minutes
Integer	Integer	4 byte integral number
DateTime	DateTime	Date and time down to and including milliseconds
WideString	WideString	Unicode string up to 255 characters
WideMemo	WideMemo	Unicode string of arbitrary length
LargeInt	LargeInt	8 byte integral number
GUID	GUID	A 16 byte globally unique identifier

The native types are documented in the TurboDB Engine documentation.

1.8.32 TurboDBLockType Enumeration

Feature supported only in TurboDB Win32 for ADO.NET

Indicates the kind of lock applied on a database table.

C#

```
[Serializable]
public enum TurboDBLockType
```

Members

Read	A read lock prevents other sessions from modifying the database table. This includes appending new rows. Any number of read locks can be placed on a table from any number of sessions at the same time.
Write	A write lock prevents other sessions from accessing the database table in any way. A write lock cannot be placed on a table, if there is already a write lock or read lock set by another session.

Compatibility

TurboDB Managed does not support this type because it has no locking capabilities.

Requirements for TurboDB Native

Namespace: DataWeb.TurboDB
Platforms: Windows 98, Windows ME, Windows NT 4, Windows 2000, Windows XP
Version Information: .NET Framework 1.1, 2.0
Assembly: DataWeb.TurboDB (for .NET Framework 1.1) or DataWeb.TurboDB.Native20.Provider (for .NET Framework 2.0)

See also

Shared Tables | Table Locks

Index

- D -

- E -

- F -

- G -

- H -

- I -

- J -

- K -

- L -

- M -

- N -

- O -

- P -

- Q -

- R -

- S -

- T -

- U -

- V -

- W -

- Y -