

Transbase®
Release Notes

Transaction Software GmbH
Willy-Brandt-Allee 2
D-81829 München
Germany
Phone: +49-89-62709-0
Fax: +49-89-62709-11
Email: info@transaction.de
<http://www.transaction.de>

Version 6.8.1.40
November 02, 2010

Contents

Version 6.8.1	3
SQL Extensions	3
UPDATE FROM statement	3
BIGINT datatype	3
Remote Views	4
Sliding Aggregates	4
Performance Improvements	4
Internal page organization	4
Multithreaded Query Evaluation	4
JDBC Driver	4
Architectural Changes	5
Delta Management	5
Version 6.7.1	6
SQL Extensions	6
Internal Keys	6
Flat Tables	6
Bitmap Indexes	7
Table Defragmentation	7
MERGE INTO statement	7
NUMBER datatype	7
Window Functions	8
Data import	8

<i>CONTENTS</i>	3
Bulk Loading	8
SPOOLing XML	9
FILE Tables	9
JDBCReader	9
OraReader	10
Performance Improvements	11
Multithreaded Query Evaluation	11
Transaction Processing	12
Architectural Changes	12
Transbase Call Interface (TCI)	12
Autocommit	13
Incremental ROM Files	13
Windows Vista	14
Version 6.4.1	15
tbadmin library	15
Hierarchical Queries in SQL	15
SQL extension	15
ORDER BY	15
ORDER BY	15
TRUNCATE	16
Query Profiling	16
Version 6.2.1	17
Large Filesystem Support	17
Differential Dump and Recovery	17
SQL extension	17
tbmode codepage <codepagespec>	17
spool codepage <codepagespec>	18
LOAD cluster <i> default	18
ROUND	18
SYSKEY	18

FIRST	18
RANK	18
UPDATE t SET (f1,f2,f3)= (SELECT x1,x2,x3 FROM ...)	19
CREATE OR REPLACE VIEW	19
SELECT <expr1> [,<expr2> ...]	19
Batch Mode	19
Tool Improvements	19
tbdiff ... -bfsize <size>	19
tbtar -r ... locale=<localespec>	19
tbarc -w ... cp=<codepagepec>	19
Adjusted Limits	20
New Tools And Interfaces	20
UTBI	20
Version 6.1.2	21
Automatic space optimization for B-trees	21
Version 6.1.1	22
Distributed Queries	22
Version 5.5.2.6	23
Database Encryption	23
Version 5.5.2.1	24
Long Filenames	24
Evaluation Plan Support	24
Administration Password	24
Library support for Backup Utilities	24
TBTar Library	24
TBArc Library	25
JAVA Stored Procedures	25
Database Triggers	25

Version 5.3.1	27
Transbase SQL extensions	27
SEQUENCE	27
Secondary Indexes	28
HyperCube Indexes	29
Tuple Construction	29
ORACLE Compatibility	29
Optimizations	30
UNICODE Databases	30
tadmin	31
SQL	31
CHAR(N), VARCHAR(N), SIZE OF	31
String functions	32
String literals	32
Spool files	32
Incremental ROM files for TransbaseCD	33
Administration	33
Case Insensitive Databases	33
TBADMIN functions	34
TBMUX: Crypted Communication	34
TCI: new programming interface	34

Version 6.8.1

SQL Extensions

UPDATE FROM statement

A new UPDATE statement has been introduced which can be used to adjust an existing table in the database with the result of a SELECT statement.

This statement will apply only minimal changes to the physical layout of the table, as opposed to the conventional tbdiff procedure. Thereby it can be ideally combined with Transbase CD for generating minimal delta ROM files.

The syntax of this command is: UPDATE <table> FROM <SELECT_Statement>

After the UPDATE statement is successfully performed the specified table contains exactly the records retrieved from the <SELECT_Statement>.

The UPDATE statement can be combined with remote access features of Transbase, e.g. Transbase D, table functions JDBCReader() and OraReader(), or direct access to flat files. The following examples illustrate some possibilities:

```
UPDATE t FROM t@sourcedb;
```

```
UPDATE t FROM SELECT * FROM FUNCTION  
  OraReader('//orasrv/oradb', 'scott', 'tiger', 'SELECT * FROM t');
```

```
CREATE FILE ('t.sp1') TABLE tsrc (<Columns_description>);  
UPDATE t FROM tsrc;
```

BIGINT datatype

Version 6.8 supports a new data type BIGINT. This type allows integer values with a size of 64 bits (including the sign).

Remote Views

A view definition may contain table names, referring to a tables residing in a remote database, accessible over Transbase D. When evaluating remote views, the privileges of the view owner apply for accessing remote tables. The current user must have at least ACCESS privilege for the remote database.

```
CREATE OR REPLACE VIEW r AS SELECT a,b,c FROM t@remotedb;
```

Sliding Aggregates

In Version 6.7, Window functions were introduced in Transbase. Now they have been extended offering powerful analytic capabilities including sliding windows with constant and dynamic offset, specified as absolute row offsets or as relative value offsets.

Performance Improvements

Internal page organization

The number of replacements of tuples in the leaf nodes of the B-Tree within one page caused by inserts, updates, and deletes are now decreased. This leads to a higher performance of writing transactions of data bases with logging mechanisms. Furthermore, the delta iteration process of Transbase CD data bases is optimized.

Multithreaded Query Evaluation

The overall scalability of parallel query processing which was introduced in version 6.7 was improved. Synchronization primitives have been optimized for highly contagious tasks with many parallel threads of execution. The parallelization of very short running queries is more graceful.

JDBC Driver

The Transbase JDBC driver was restructured now yielding a significantly reduced code basis and improved performance. JDBC and JDBCX modules were repackaged into one centralized jar file. This jar is now available in two versions, as faster release build without debug and logging capabilities and as separate debug build with logging and additional debug information.

Architectural Changes

Delta Management

Delta Management deals with production and delivery of updates referring to a retrieval database stored on CD/DVD. Up to Transbase Verion 6.7, updates could be realized by delta romfiles which can be atached togetherwith the original romfiles at client site.

With Transbase Verion 6.8, so called "transitional romfiles" realize a new update technique which avoids the drawback of increasing data transfer when updates are produced and delivered iteratively. Thereby, client data updates can be realized in higher frequency and more economically.

Version 6.7.1

SQL Extensions

Internal Keys

In previous Transbase versions tables were typically created with internal key (IK) access path. This IK access path requires additional space of 6 to 8 bytes per tuple. It is used as row identifier, e.g. for referencing tuples in the base relation after accessing secondary indexes. Tables could be created explicitly `WITHOUT IKACCESS` to save space, but this inhibited the creation of secondary indexes on these tables.

Now it is possible to create secondary indexes on IK-less tables. In this case the base table's primary key is stored in all index tuples for referencing the base relation. This leads to a notable space reduction in many cases.

As the primary key can be rather extensive, Transbase will decide at table creation time whether to create a table `WITH` or `WITHOUT IKACCESS`. This guarantees optimal space efficiency. If the primary key occupies no more than the IK, then the table is created `WITHOUT IKACCESS`. Else an IK access path is added by default.

It is always possible to override this default mechanism by adding `WITH` or `WITHOUT IKACCESS` to the create table statement.

Flat Tables

Flat tables are a new form of base relations. They store data in input order and do not have a primary key. Therefore they allow faster data loading. If a flat table was created `WITH IKACCESS` it is possible to add secondary indexes for efficient lookups. Typically secondary indexes are added once the load process is complete, so load performance is not compromised.

In addition, Flat Tables can be restricted to occupy no more than a certain maximum of space. If this maximum is exceeded, the oldest data will be replaced.

Thus Flat Tables are ideally suited for data staging during bulk load processes, as temporary storage and for logging facilities.

For details see the Transbase SQL Guide.

Bitmap Indexes

Bitmap indexes have been introduced as a new form of secondary indexes. They are preferably used for non-selective columns having few different values (e.g. classifications). Bitmap indexes are innately very space efficient. With their additional compression in average they occupy less than one bit per index row. A bitmap index can be created on any base relation (B-Tree or Flat) having a single INTEGER field as primary key or an IKACCESS path.

Bitmap processing allows inexpensive calculation of logical combinations (AND/OR/ NOT) of restrictions on multiple non-selective fields using bitmap intersection and unification.

For details see the Transbase SQL Guide.

Table Defragmentation

Tables receiving many random INSERTs, UPDATEs or DELETEs suffer a fragmentation of their logical page ordering. This can slow down sequential scans significantly.

By using the ALTER TABLE . . . MOVE statement those tables can be reorganized with optimal page adjacency. During this process the database remains online and accessible.

For details see the Transbase SQL Guide.

MERGE INTO statement

This new SQL command allows tuples to be inserted or updated within existing tables depending on user-specified conditions.

For details see the Transbase SQL Guide.

NUMBER datatype

In previous Transbase versions the datatype NUMERIC(*p*, *s*), *s* and *p* had to satisfy the equation $0 \leq s \leq p \leq 30$; it covered numbers having at least one and at most 30 digits and optionally up to *s* digits after the decimal point.

This datatype NUMERIC(*p*, *s*) has been generalized as follows:

For the precision $0 \leq p \leq 30$ still holds; NUMERICs with precision 0 are always 0, independently of their scale.

The scale *s* is restricted to the range $-127 \leq s \leq 127$.

A positive scale means the maximum number of digits after the decimal point. A negative scale analogously means a representation in tens, hundreds, thousands etc.

(*p,s*) may be omitted completely (denoted simply by `NUMBER` or `NUMERIC(*,*)`) to specify numbers having at most 30 digits with arbitrary scale.

The arithmetic routines of `NUMERIC` automatically round results to the maximum precision of 30 digits if necessary. Overflow happens if the scale becomes greater than 127 or less than -127.

Window Functions

Window functions have been introduced as extension to the SQL standard aggregation functions. They allow aggregation over a sliding window on a result set. This offers the possibility to calculate running totals, sliding averages and rankings over record sets. All aggregation functions (`AVG`, `COUNT`, `MAX`, `MIN`, `SUM`) are supported and additional `RANK` and `DENSE_RANK` functions have been added. The syntax and semantics follow the SQL 2003 specification.

The calculation of a running total and sliding average per `deptno` is expressed in the following

Example:

```
SELECT deptno,  
       SUM(salary) OVER (PARTITION BY deptno ORDER BY emp_id),  
       AVG(salary) OVER (PARTITION BY deptno ORDER BY emp_id)  
FROM employees
```

The details on this feature can be found in the Transbase SQL Guide.

Data import

Bulk Loading

The implementation of Transbase bulk loading via the `SPOOL` command has been improved:

The input data is first sorted on the primary key of the destination table (if necessary), then the primary table is built. This guarantees optimal adjacency of base relation tuples. In parallel, for each secondary index, the tuples are stored temporarily.

When the primary table is built completely, each secondary index is built up one by one:

- sort tuples into secondary key order
- build up the index

As a result, the adjacency of pages is optimal for sequential scans on the table and all indexes.

SPOOLing XML

The Transbase `SPOOL` command has been modified to support both delimiter separated files and XML structured files.

For details see the Transbase SQL Guide.

FILE Tables

Data stored in files may now be integrated into the database schema as virtual tables. These `FILE` tables offer read-only access to those files via SQL commands. They can be used throughout SQL `SELECT` statements like any other base relation.

Example:

```
CREATE FILE (/usr/temp/data.csv)
  TABLE file_table WITHOUT IKACCESS
  (a INTEGER, b CHAR(*))

SELECT a+10, upper(b) from file_table
SELECT b FROM file_table, regular_table
  WHERE file_table.a=regular_table.a
```

`FILE` tables are primarily designed as an advanced instrument for bulk loading data into Transbase and applying arbitrary SQL transformations at the same time.

For details see the Transbase SQL Guide.

JDBCReader

A built-in table function `JDBCReader` provides read-only access to arbitrary JDBC data sources. The function may be used throughout SQL `SELECT` statements like any other base relation.

It requires four `CHAR(*)` parameters.

- a JDBC connection string, i.e.

- ```
'jdbc:protocolname://hostname[:port][/dbname]'
```
- the user name,
  - the password,
  - an arbitrary SQL SELECT query, e.g.
- ```
'SELECT x,y FROM S,T where S.a=T.a AND s.b IS NOT NULL'
```

It can be used for querying remote JDBC data sources or for data import.

Example:

```
INSERT INTO T SELECT * FROM
  FUNCTION JDBCReader('conn_string','user','passwd',
    'SELECT * FROM jdbc_table')
```

The data retrieved from the external data source is converted into Transbase datatypes for further processing. Data types are mapped appropriately. Data truncation may happen at this point if the target data types is smaller in range. A few data types cannot be mapped; those data types should be converted on the remote site into mappable data types. BLOB data is currently not supported.

The appropriate JDBC driver for the designated JDBC data source has to be provided.

OraReader

Similar to the `JDBCReader`, the `OraReader` provides read-only access to remote Oracle databases. For maximum efficiency, the function is implemented via a dynamic link library (in C programming language) using the OCI interface to access Oracle. Thus it will outperform the `JDBCReader` on Oracle data sources. The function may be used throughout SQL SELECT statements just like any other base relation.

It requires four CHAR parameters:

- the name of the data source, e.g.
- ```
'//hostname:[port][/servicename]'
```
- the user name,
  - the password,
  - an arbitrary Oracle SELECT query, e.g. 'SELECT \* FROM ora\_table'

It can be used for querying remote Oracle data sources for data import.

**Example:**

```
INSERT INTO T SELECT * FROM
 FUNCTION OraReader('//orasrv/oradb','scott','tiger',
 'SELECT * FROM ora_table')
```

The data retrieved from Oracle is converted into Transbase datatypes for further processing. A few Oracle data types cannot be mapped; those data types should be converted within Oracle into mappable data types. In contrast to the `JDBCReader`, the `OraReader` also supports BLOB data.

An Oracle client installation is required on the Transbase host. Because of its ease of handling, the Oracle Instant client is recommended over the standard Oracle client installation for this purpose. The software is freely available from Oracle. Its version should match the version of the Oracle database server. On Windows Platforms, please make sure that the `PATH` environment variable includes the Oracle client before the Transbase service is started. For Unix platforms, the `LD_LIBRARY_PATH` variable must be set and include the Oracle client before the Transbase service is started. Also other Oracle environment variables should be set before Transbase is started, including `ORACLE_HOME` etc.

## Performance Improvements

### Multithreaded Query Evaluation

Transbase will split the calculation of complex and long running queries into several tasks and run these tasks in parallel using multiple threads of execution. This feature might also improve queries on single processor machines, but it is primarily intended for multiple-CPU computers.

By default, parallelization is switched off. It can be activated for a database with the `tbadmin` tool, by setting an option to the desired level of parallelization. The options are:

- `mt=max` activates the full potential of multithreading; it establishes data pipelines in query plans that run in parallel, also using out-of-order execution, for improved overall performance.
- `mt=det` offers fair parallelism while producing result sets in deterministic output order. Performance is likely to suffer somewhat compared to maximum parallelism, as data pipelines operate only in first-in, first out mode.
- `mt=min` is a rather defensive strategy of parallel query execution; parallel execution is limited to I/O relevant nodes (e.g. `REL` or `REMOTE`) and activates work-ahead for the entire `SELECT` query,

- `mt=off` means no parallelization at all (default),

Alternatively the setting can be manipulated on session level using a `TBMODE` statement similar to the `tbadmin` command:

```
TBMODE MULTITHREAD {MAX | DETERMINISTIC | MIN | OFF}
```

**Note:** Without multithreading Transbase guarantees that data is always processed and returned in the same deterministic sort order, even if no `ORDER BY` was specified. I.e. the same query produced always the same result in the same order. The SQL specification does not demand any reproducible sort order if no `ORDER BY` is used. With multithreaded query processing switched to `MAX` it now likely that data is processed out-of-order. Thus a query will return the same result but possibly in different order if no `ORDER BY` is specified. Only the specification of an `ORDER BY` guarantees a result sort order.

For supporting legacy applications while still enabling multithreaded query execution Transbase supports multithreading in `DETERMINISTIC` mode. This mode offers a fair degree of parallelism while conserving the behaviour of single threaded query evaluation.

## Transaction Processing

The transaction processing in multi-user environments has been improved by minimizing the time while the database is exclusively locked at the end of a transaction.

In particular, the I/O phases needed to write through committed pages or to write back aborted pages into the diskfile are non-exclusive.

## Architectural Changes

### Transbase Call Interface (TCI)

The TCI interface is introduced to replace the TBX interface. TCI is thread safe and uses handles to internally synchronized data structures. Handles are provided in an object-oriented manner, e.g. for

- environment objects,
- database connection objects,
- transaction objects,
- statement objects,

- query result objects,
- error objects.

TCI is completely described in a separate manual Transbase TCI guide.

Most of the database driver such as: ODBC, OleDb, .net, PHP internally use the TCI interface. Therefore their implementation is also thread safe.

### **Autocommit**

The Transbase kernel has been extended to provide an autocommit mode for transaction processing. This means that applications do not need to control transactions explicitly. Each cursor is processed in its own (separated) transaction which is closed with the cursor.

In particular, INSERT, DELETE and UPDATE statements are committed immediately.

Autocommit transactions are provided by TCI by setting environment or database properties. Driver APIs (including JDBC) use this kernel feature to implement autocommit transactions.

### **Incremental ROM Files**

The command `tbmkrrom` has been modified to produce incremental ROM files in a more compact form; in former versions, a modified page was compressed and then provided as part of the incremental ROM file. At client side, the page in the incremental ROM file replaced the original page of the ROM file.

This mechanism can be improved for pages that receive only minimal changes (e.g. a single tuple or a single value has changed). In such cases, a so-called XOR page is computed which compresses much better. At client side, XOR pages can be used to re-construct the modified page by applying this XOR page to the original page. Depending on the compressed XOR size, the decision is made for each page whether XOR pages are used or complete pages.

While this technique may save space significantly, it makes page access to modified pages a little bit slower at client side.

The command syntax has been extended as follows:

```
tbmkrrom [-f] <database> [incremental [xor]]
```

## Windows Vista

The Transbase setup procedure has been modified for Windows Vista. In particular, Transbase installs into two different directories `c:\Program Files\Transbase` and `c:\ProgramData\Transbase`.

Program files that are read-only are placed in the first directory, while other installation files (such as `dblist.ini`) which are read and written are placed in the second directory. Make sure that the second directory provides read and write access to all Transbase users.

In addition, all Transbase users must be granted the explicit privilege to create and access global shared memories. We recommend that this privilege should be granted to a group which every Transbase user should belong to.

See details in our technical note on Windows Vista, which contains particular advice for older Transbase versions, too.

# Version 6.4.1

## tbadmin library

dump and restore functionality has been added to tbadmin library.

## Hierarchical Queries in SQL

In addition to the *SearchCondition* in the WHERE clause hierarchical data can be queried using *HierarchicalSearchConditions*. Here a depth-first search is carried out starting with one of the root rows that satisfies the START WITH predicate. For this root row the first child rows satisfying the CONNECT BY condition is selected. Then the hierarchical search will proceed down through the generations of child rows until no more matching rows are found.

## SQL extension

### ORDER BY

ORDER BY accepts expressions. The elements in an ORDER BY are no more restricted to simple fields but may contain any expressions composed of constants, fields, functions and subqueries. The denoted fields must have a resolution either against fields in the corresponding SELECT list or to fields resolvable by the corresponding FROM clause or (if occurring in subqueries) by the surrounding query blocks.

### ORDER BY

ORDER BY is accepted in inner blocks now. An ORDER BY in a subquery of a query syntactically is accepted. However, it is not guaranteed that such an ORDER BY has an effect on the ordering of the overall query result.

## TRUNCATE

TRUNCATE <table> is equivalent to DELETE FROM <table>.

## Query Profiling

Support for query evaluation plans including sampling of elapsed times can be enabled by a `tbmode` statement. Support is enabled by `tbmode profiles on` and disabled by `tbmode profiles off`.

# Version 6.2.1

## Large Filesystem Support

Transbase now supports large filesystems. Actually LFS is supported on **Linux Solaris** and **Windows 2K/XP/2003** platforms. Only database diskfiles may be configured bigger than 2GB, all other files still are limited by 2GB.

The maximum diskfile size is limited by the filesystem.

## Differential Dump and Recovery

Alternatively to the Full Dump a Differential Dump and Recovery method is provided. For differential dumping an initial dump consisting of diskfiles and logfiles is required. Afterwards only logfile changes since the last dump are moved to a safe place, making dumping in short intervals swifter. The database remains operational while the dump is performed.

The following command produces an initial dump into a directory 'dumpdir':

```
tbadmin -drec dump db=<dbname> dir=<dumpdir>
```

Whereas a differential dump is made with:

```
tbadmin -drec dump diff[erential] db=<dbname> dir=<dumpdir>
```

## SQL extension

**tbmode codepage <codepagespec>**

sets default codepage for spool statements. <codepagespec> is one of:

|          |        |        |                                   |
|----------|--------|--------|-----------------------------------|
| PROPRIET | UTF8   | SJIS   | EUC                               |
| UCS      | UCS2   | UCS2LE | UCS2BE                            |
| UCS4     | UCS4LE | UCS4BE | 'locale string (valid on server)' |

where UCS/UCS2 are mapped to UCS2LE and UCS4 is mapped to UCS4LE.

**spool .... codepage <codepagespec>**

more codepage conversions supported. <codepagespec> is one of:

|          |        |        |                                   |
|----------|--------|--------|-----------------------------------|
| PROPRIET | UTF8   | SJIS   | EUC                               |
| UCS      | UCS2   | UCS2LE | UCS2BE                            |
| UCS4     | UCS4LE | UCS4BE | 'locale string (valid on server)' |

where UCS, UCS2 and UCS4 to the server supported little or big endian.

**LOAD cluster <i> default**

Loads all tables in cluster <i> with default algorithm. In cases where romfiles reside on CD-ROM it is useful to have a DEFAULT LOADING which is restricted to all tables/indexes of a certain cluster. This command replaces a sequence of LOAD commands for all tables/indexes of a cluster.

**ROUND**

A ROUND function is provided in the form ROUND( <expression> [ , <scale> ) which makes a arithmetic rounding of the arithmetic value to the desired scale. This equivalent to a CAST function to a NUMERIC value with the desired scale.

**SYSKEY**

Result tuples of a non-join query may contain the internal key provided the corresponding table has been created with the IKACCESS option. The syntax is SYSKEY which may appear in the outermost SELECT list. The application subsequently can make a fast direct access to the corresponding table by 'SELECT .. FROM <table> where SYSKEY = <syskeyvalue> The type of the SYSKEY field is CHAR(\*)'.

**FIRST**

A SELECT query may contain a FIRST clause at the end to limit the result tuples to a desired number. For example, to limit the result to 3 tuples, the syntax is 'SELECT ... FROM ... WHERE ... FIRST(3)'

**RANK**

The analytic function RANK is provided. It may appear inside a SELECT list and produces the ordinal number of the result tuple of the query block according to a specified ORDER BY inside the RANK specification. Additional partitioning of the ORDERING may be specified.

**UPDATE t SET (f1,f2,f3)= (SELECT x1,x2,x3 FROM ...)**

The SET clause of an UPDATE statement may contain bracketed field sequences which obtain their values by a corresponding matching n-ary subquery.

## CREATE OR REPLACE VIEW

'CREATE OR REPLACE <viewname>' is now supported. If a view 'viewname' already exists and the new view definition is correct, then the old view definition is 'overwritten' by the new one.

**SELECT <expr1> [,<expr2> ...]**

Queries may consist of a query block which only contains one expression or a sequence of expressions without FROM clause. This is equivalent to the dummy table 'dual' used by other systems. For example, to get the current user of the current connection, a query 'SELECT USER' is appropriate.

## Batch Mode

TCI and JDBC support batch mode stored/prepared DML statements. A stored INSERT/UPDATE/DELETE statement with dynamic parameters '?' can be run several times with a set of different parameters. The parameters are cached on the client side and transferred to the server in one or few large packages when the batch is executed. This significantly reduces communications overhead.

## Tool Improvements

**tbdiff ... -bfsize <size>**

tbdiff accepts option '-bfsize <size>' for sampling several blobs within one file.

**tbtar -r ... locale=<localespec>**

tbtar supports option 'locale=' for codeconversion when loading databases.

**tbarc -w ... cp=<codepagepec>**

tbarc supports option 'cp=' for codeconversion when archiving databases.

## **Adjusted Limits**

Up to 64 sessions and up to 64 transactions for one application are accepted now. The maximum tuplesize has been increased from 4kb - 96b bytes to 32kb - 96b. The maximum tuplesize is limited to the database's pagesize.

## **New Tools And Interfaces**

### **UTBI**

Transbase now comes with a new commandline interface UTBI. This application supports unicode data.

# Version 6.1.2

## Automatic space optimization for B-trees

table columns are automatically arranged in a space optimal order. Manual optimization on DDL level is not longer needed.

# Version 6.1.1

## Distributed Queries

Version 6 introduces the concept of distributed query. An application connected to one single database might reference a table on a remote database by suffixing the SQL table name with @dbname@hostname. Distributed joins can be run with that mechanism. Also INSERT statements (without BLOBs) can be processed against remote tables. BLOBs, however, can also be fetched via SELECT queries from remote tables.

DDL and VIEW statements against remote tables are not allowed. Also BLOB fields can only be fetched from local tables.

See details in [distributed queries](#).

# Version 5.5.2.6

## Database Encryption

Transbase now supports database encryption. Database encryption must be enabled by using `tbadmin`.

# Version 5.5.2.1

## Long Filenames

The files 'f.db' and 'f.bak' have been replaced by 'dblist.ini'. The format has been changed to an INI-format which allows handling long filenames and filenames containing spaces.

The database description file 'tbdesc' has been replaced by 'dbconf.ini'.

## Evaluation Plan Support

Support for evaluation plans can be enabled by a `tbmode`-statement. Support is enabled by `tbmode plans on` and disabled by `tbmode plans off`. Plans can be retrieved on `tbx`-level by calling `tbx` (`TB_GETPLAN, ...`).

`tbi` supports Evaluation Plans.

## Administration Password

An administration password has been introduced for creating, deleting and administering databases. As a consequence `tbadmin-password` is no longer needed for deleting a database.

## Library support for Backup Utilities

### TBTar Library

A library is supplied for developing `tbtar` compliant applications. A more detailed description is given in TBTar SDK

## TBArc Library

A library is supplied for developing tbarc compliant applications. A more detailed description is given in TBArc SDK

## JAVA Stored Procedures

User defined JAVA programs can be installed in the database. 3 different types of JAVA programs are distinguished.

A so called "Stored Procedure" is a JAVA procedure which is called by a "CALL proc(..)" statement on the TBX interface using the TbxRun function. If it has OUT parameters, it also delivers one result tuple which can be interpreted by the application exactly like a TbxRun on a SELECT statement.

A so called "Table Function" is a JAVA procedure with OUT parameters which can deliver a tuple set. It can be used in the FROM clause of a SQL query block like any other basetable or view.

A so called "User defined function" is a JAVA function which returns exactly one value at each call. It can be used in an arbitrary SQL DML statement at each place where an expression (delivering exactly one value) is accepted (i.e. SELECT, WHERE and HAVING clause).

Each of the above types may have arbitrary IN parameters. Actual parameters may be SQL expressions inside a SQL statement or dynamic parameters '?' which then are supplied by the application. In the latter case, the statement has to be stored by TbxStore like any other parameterized SQL statement.

Each JAVA program may itself open cursors to the database via the JDBC interface and may have side effects on database tables.

## Database Triggers

Database Triggers can be created by CREATE TRIGGER .. and dropped by DROP TRIGGER .. Functionality and syntax is closely related to the SQL3 standard.

Triggers can be specified as tuplewise or statement related.

For each trigger, a database event is specified which fires the trigger. Database events are INSERT, DELETE and UPDATE. UPDATE can be restricted to a subset of fields of the table.

Optionally, a SQL predicate (search condition) can be specified to restrict the firing of the trigger to a value dependent condition.

Firetime of the trigger can be specified as BEFORE or AFTER the database event.

The actions of a trigger can be one or several DML statements such as INSERT, UPDATE, DELETE. Also the execution of a JAVA stored procedure (CALL javaproc(..) ) can be an action.

# Version 5.3.1

## Transbase SQL extensions

### SEQUENCE

A new construct `SEQUENCE` has been introduced which serves to generate database-wide unique numbers. A sequence is constructed with a user defined name. A start value and an increment value can optionally be specified. Per default a sequence starts with value 1 and increments by 1. A sequence with name `S` provides two functions, namely `S.nextval` to generate and deliver a new number and `S.currval` to refer to the most recently delivered number within the current application.

An existing `SEQUENCE` can be used as part of a `DEFAULT` clause for a field of a table. In this way, a field serves as an automatically generated synthetic key.

#### Example:

```
CREATE SEQUENCE si START 0;
CREATE SEQUENCE sii START 0;

CREATE TABLE invoice (
 n INTEGER DEFAULT si.NEXTVAL,
 custid DECIMAL(5),
) KEY IS n

CREATE TABLE invoice_item(
 n INTEGER REFERENCES invoice(N),
 item INTEGER DEFAULT sii.NEXTVAL,
 description VARCHAR(80),
 price DECIMAL(12,2),
 ...
) KEY IS n, item;
```

The following statements could be used to generate a new invoice with two items.

```
INSERT INTO invoice (n, custid)
VALUES (si.NEXTVAL, 12345);
```

```
INSERT INTO invoice_item(n, item, description, price)
VALUES (si.CURRVAL, sii.NEXTVAL, 'First', 33.33);
```

```
INSERT INTO invoice_item(n, item, description, price)
VALUES (si.CURRVAL, sii.NEXTVAL, 'Second', 123.45);
```

## Secondary Indexes

In secondary indexes, participating fields and secondary keys now can be specified separately.

For example, assume a table with fields f1, f2, f3, f4 where f1 is the key:

```
CREATE TABLE f (
 f1 INTEGER,
 f2 INTEGER,
 f3 INTEGER,
 f4 INTEGER)
KEY IS f1 ;
```

Analogously, it is now possible to specify a KEY clause for secondary indexes that implies an automatic uniqueness restriction. So, the following two notations are equivalent:

```
CREATE UNIQUE INDEX findex ON f (f2, f3, f4)
```

```
CREATE INDEX findex ON f (f2, f3, f4) KEY IS f2, f3, f4
```

The semantic separation between index fields and unique-key fields, however, allows constructs like:

```
CREATE INDEX findex ON f (f2, f3, f4) KEY IS f2, f3
```

where the uniqueness restriction is given by (f2, f3) only. Although a UNIQUE INDEX on (f2, f3) could have been defined, it may be very useful to have additional fields in a secondary index in order to omit the base tuple materialization. The query

```
SELECT f2, f4 FROM f WHERE f2=5
```

would need to access the base tuples when an index is defined only on (f2, f3) while it could be resolved solely from the secondary index when an index contains (f2, f3, f4).

## HyperCube Indexes

For Transbase HyperCube, secondary indexes can be built on the numeric data types: TINYINT, SMALLINT, INTEGER and NUMERIC(P,S).

### Example:

```
CREATE TABLE points (
 id INTEGER,
 x NUMERIC (10,7) NOT NULL CHECK(x BETWEEN -180 AND 180),
 y NUMERIC (10,7) NOT NULL CHECK(x BETWEEN -90 AND 90),
 info CHAR(*),
 ...)
KEY IS id ;
```

```
CREATE INDEX xpoints ON points(x,y,info) HCKEY IS x,y ;
```

The secondary index contains the fields x, y, and info. The hypercube key is defined on x and y only. Note that this example uses the new extended syntax for secondary indexes described above.

All fields used as hypercube key in the secondary index must be of appropriate type and be specified as NOT NULL and have a CHECK CONSTRAINT with range condition in the underlying table definition.

## Tuple Construction

For SQL2 conformity, tuples can be constructed by parentheses as well as by brackets, e.g.

```
where (tuple) in (select ...).
```

## ORACLE Compatibility

Various ORACLE functions have been resembled with identical semantics and syntax, including SYSDATE, DECODE, TO\_CHAR, NVL, TRUNC in order to alleviate the migration from ORACLE to Transbase.

## Optimizations

Various optimizations improve the computation of predicates on B-tree layer. Projections are handled by the B-tree layer, too, in order to save space and time of processing.

Space requirements of operator trees have been reduced for joins.

## UNICODE Databases

The UNICODE standard maps characters of all existing languages into a single coding schema. Characters are either denoted by 16-bit values (UCS-2) or by 32-bit values (UCS-4). For databases, however, to store each character as fixed 2 or even 4 bytes, would be too inefficient. Instead, Transbase uses the UTF-8 coding of UNICODE characters to store them on disk and to send and receive them over communication lines.

The table below shows valid UTF-8 codings:

|          |          |          |          |          |          |         |
|----------|----------|----------|----------|----------|----------|---------|
| 0xxxxxxx |          |          |          |          |          | 7 bits  |
| 110xxxxx | 10xxxxxx |          |          |          |          | 11 bits |
| 1110xxxx | 10xxxxxx | 10xxxxxx |          |          |          | 16 bits |
| 11110xxx | 10xxxxxx | 10xxxxxx | 10xxxxxx |          |          | 21 bits |
| 111110xx | 10xxxxxx | 10xxxxxx | 10xxxxxx | 10xxxxxx |          | 26 bits |
| 1111110x | 10xxxxxx | 10xxxxxx | 10xxxxxx | 10xxxxxx | 10xxxxxx | 31 bits |

In particular, it can be seen that

- ASCII characters in the range of 0..127 are mapped into themselves;
- single-byte characters in the range 128..255 (i.e. those in the Latin-1 charset) are mapped into a sequence of two bytes;
- other UCS-2 characters are mapped into sequences of at most three bytes;
- UCS-4 characters are mapped into sequences of at most six bytes.

It can be seen, too, that

- not every sequence of 8-bit bytes is a valid UTF-8 coded string,
- the mapping is only unique if characters are mapped into sequences of minimal length and
- the sort order is preserved by the coding.

The fact that a single character may consist of more than one byte, influences the string functions and string predicates. The `SUBSTRING` function e.g. assumes its parameters to be characters (not bytes) and the `LIKE` predicate expects an underscore to denote a single character (not byte).

Please note that the definition of fixed length characters like `CHAR(N)` still is given in bytes (not in characters) because a length of 2 characters would result in a (non-fixed) bytelength of between 2 and 12. A length restriction in characters could be formulated as: `CONSTRAINT (CHARACTER_LENGTH(x) = 2)`.

At the communication layer, UTF-8 is used, too. In particular, TBX strings are assumed to be valid UTF-8, both on input (queries, parameters) and on output (result strings). UTF-8 should be mapped into either `UNICODE` or into an appropriate single-byte charset by the application.

For conversion of single-byte databases, it is recommended to export the data, convert it into UTF-8 or `UNICODE` and then to import it. The Transbase Spooler has been extended to accept single-byte, UTF-8 and `UNICODE` input.

In particular, the following important modifications apply for Unicode databases.

## **tadmin**

The coding of databases has to be specified upon creation. It cannot be changed later. Code pages can be: proprietary (upward-compatibel to former Transbase versions), ASCII (only ASCII characters below 128 are permitted), single-byte (arbitrary single-byte characters are permitted), and UTF-8 (arbitrary Unicode characters are permitted).

Along with the coding, a fixed locale setting can be defined which influences national character processing.

## **SQL**

### **CHAR(N), VARCHAR(N), SIZE OF**

When defining fixed-length strings, a maximum number of bytes has to be specified rather than a maximum number of characters. The reason for this decision is that the specification of `N` is space-related. In particular, `CHAR(5)` strings must have fixed length of 5 bytes, while strings of five characters may vary in size between 5 and 15 bytes(for UCS-2).

A size restriction for a field `COL` in terms of characters can be specified by an additional `CHECK CONSTRAINT (CHARACTER_LENGTH(COL) < 10)`.

In contrast, the `SIZE OF` operator for strings returns the number of bytes taken by the UTF-8 representation.

### String functions

All string related functions work in terms of characters (not bytes).

In particular, the function `CHARACTER_LENGTH(x)` returns the number of Unicode characters in `x`, while `SIZE OF x` returns the number of bytes.

The function `SUBSTRING(x from 1 for 2)` returns the first two characters of `x`. `SIZE OF (SUBSTRING(x from 1 for 2))` may return values between 0 and 6 for UCS-2 strings.

### String literals

String literals containing UNICODE characters can be constructed by either of the following methods:

```
'Mnchen'
```

```
'M' + 0u00fc + 'nchen'
```

```
'M' + 0xc3bc + 'nchen'
```

The first method requires the ability to type the unicode character directly (which may not be possible in any environment).

The second method uses a new string literal variant which denote Unicode characters by their hexadecimal value. Note that always four hexadecimal digits must be specified per Unicode character. Thereby, more than one Unicode character representation can be concatenated, e.g. `0u00fc006e`.

The third method provides a direct UTF-8 representation (syntactically a `BINARY`) of that part of the string that cannot be expressed literally. Again note, that for this representation a valid UTF-8 coding is required.

### Spool files

When data is loaded from or extracted into external files by the Transbase `SPOOL` command, it has to be specified in which format the file should be coded.

For this purpose, the syntax of the `SPOOL` statement has been extended. See SQL guide for more details.

## Incremental ROM files for TransbaseCD

TransbaseCD offers a new space saving possibility to propagate database updates for CD-ROM databases.

To propagate updates onto a CD-ROM database (which might be distributed in many copies) two alternatives have been offered in the past:

1. Distribute SQL scripts (and possibly spool files) which are executed on CD-ROM site.
2. Accumulate all updates on one master CD-ROM database (updates then are reflected in a corresponding diskfile). Then distribute copies of the diskfile to the target sites.

The first solution is complex and requires runtime equipment on the target sites. The second solution is platform dependent and tends to produce large data volumes.

A third method is provided now: Updates are prepared as in the second solution on one master site. Then a new kind of FLUSH operation ("incremental flush") is performed on that CD-ROM database. Thereby, so-called delta romfiles are produced. These delta romfiles then are distributed to the target sites. Blocks appearing in a delta romfile shadow the blocks in the original romfiles. Delta romfiles are automatically compressed and thus are ideally suited to be distributed over the Internet. Moreover, delta romfiles become platform independent, too, like standard romfiles.

## Administration

### Case Insensitive Databases

Databases can be defined to be "case insensitive". In this case, all identifiers (field names, table names etc.) are compared by case-insensitive routines. When inserted into the data dictionary, those identifiers are mapped to capitals. This has to be considered when the data dictionary is searched explicitly by SQL queries on system tables.

Note that "delimiter identifiers" are still case sensitive.

By default, databases are case sensitive. Migrated old databases are also case sensitive. A case sensitive database can also be converted to a case insensitive database by an explicit `tadmin -a` command. In rare cases, this conversion might fail if table or field names exist which would map to the same capital letter identifier.

## **TBADMIN functions**

The cache size (global shared memory) of a database now can be set up to a maximum value of 20 Gbyte.

Temporary files possibly needed by the kernel are no longer placed in the TMP or TMPDIR directory, but in the scratch directory of the database. Therefore, it should be made sure that sufficient space is available for scratch files.

## **TBMUX: Crypted Communication**

The communication between Transbase kernel and applications is handled by TBX. The communication can be declared to be crypted in order to increase the privacy of the client/server communication.

Cryptification is specified by the command `tbmux -crypt ...` and therefore applies to all communications with this database server.

## **TCI: new programming interface**

A new programming interface, Transbase Call-Interface (TCI) has been implemented. TCI supports ANSI and UNICODE applications.