Transbase® Administration Guide

Transbase Administration Guide

Version V8.4

Publication date 2023-01-25

Table of Contents

Introduction

This manual covers the administration of Transbase databases.

This does not only include the creation, deletion and configuration of single databases but also advanced configurations tasks for cooperating databases and services:

- Creation, deletion and configuration of databases
- Backups, registration and deregistration of databases
- Database replication and the configuration of database grids
- Configuration of database crowds an crowd services
- Publication of databases and installation of publicated databases

For the configuration of Transbase services refer to the *Transbase Service Guide* [transbase_service.xhtml]

For this tutorial we use the interactive frontend of Transbase® called `tbi` which is described in detail in *Transbase® TBI Guide* [tbi.xhtml]. All examples use the simple database name `db`.

1. Databases

1.1. Private and Public Databases

There are two different kinds of databases: private and public databases.

Private Databases are administered by an embedded Transbase that is part of the application. Private databases are identified by the directory that contains all the files belonging to the database. This is just a relative or an absolute path with the file prefix.

```
e.g. file:///databases/privatedb
```

Public Databases are administered by the Transbase service on the network. They are identified by a database name and the service. The Transbase service is simply identified by the URI of its host and port.

```
e.g. //www.transaction.de:2024/publicdb
```

1.2. Database Files

Each database has its own database directory.

By default, all data of a database reside in files inside the database directory, i.e. in proper subdirectories of the database directory. At database creation time, however, other locations can be specified for different portions of the database data. The database directory contains at least a database description file, which is named `dbconf.ini`. This file contains the basic configuration of the database.

Each database directory contains the following subdirectories as containers for different types of data:

- *disks* Permanent data is stored in the logical disks container. This container contains one or more disk files. These diskfiles need not reside inside the same directory (but they do by default).

  Transbase names the files `disk00000.000`, `disk00000.001`, etc. The first number is running number, the second number gives the number of the dataspace to which the file belongs.

- *roms* The ROM files container contains the ROM files of a retrieval database and is explained in detail in the *Transbase Publishing Guide* [tbcd.xhtml].

- *bfims* ("before images") is a directory that stores data needed guarantee transactional integrity and to recover the database data after a system crash.

  Transbase supports two different logging strategies: *'before image logging'* and *'delta logging'*. Regardless of the chosen method the information needed to rollback transactions is stored in this subdirectory.

- *import* The import directory contains private subdirectories for each database user. It is used for the create file table or the spool local statement.

- *scratch* is a directory where temporary files are created as needed (intermediate results, sorting, etc.) and destroyed at the end of transactions.

- *trace* is a directory where trace output is stored by default.

There is another configuration file in the DATABASE_HOME directory of the Transbase service called `dblist.ini`. This file contains a list of all databases and their locations.

The two configuration files `dbconf.ini` and `dblist.ini` are automatically updated by Transbase.

1.3. The Repository Database `admin`

Public databases can be administrated via SQL like commands on the repository database admin. This special database is automatically installed and managed by the Transbase service.

See appendix *Administration Language* for the details of administration commands.

When a Transbase server is started an administration database called admin is created automatically. All database administration on this server is performed by connecting to the admin database with any suitable Transbase frontend or application. On this admin database, however, only database administration for this server can be performed.

The admin database may also be used to change database parameters after database creation. For some changes it may be necessary to shut down the database concerned.

```
# tbi
no database > connect //www.transaction.de:2024/admin
Login: [tbadmin]
Password:
admin >
```

2. Database Administration

In general, database administration is done via SQL like commands. It only works in AUTO-COMMIT mode, because there is no ABORT mechanism for administration tasks. Instead the opposite operation has to be done afterwards. For an example if a database was created by mistake it must be dropped afterwards by the user himself. It is not possible to abort this action. Some operations are even not reversible like the migration of databases.

```
# tbi //www.transaction.de:2024/db
Login: [tbadmin]
Password:
db-> SET autocommit on;
db > ALTER DATABASE SET recovery_method=logging;
db >
```

2.1. Create Database

This action is different for public and private databases.

Public databases are created via a SQL like command on the repository database. You just have to write *CREATE DATABASE* followed by the database name. The database is created in the databases subdirectory of the Transbase installation. It is also inserted into the dblist.ini.

```
# tbi
no database > connect //www.transaction.de:2024/admin
Login: [tbadmin]
Password:
admin > CREATE DATABASE publicdb;
admin >
```

Private databases cannot be created by an SQL command because there is no connection to the repository database. That's why private databases are created during the first connect to the database. It does not matter if it is the tbi or a self-written java application, the first connect to a private database always creates the database.

```
# tbi
no database > connect file:///mydbs/privatedb
Login: [tbadmin]
Password:
/mydbs/privatedb->
```

With the previous commands we created databases with default settings. If we want to change some settings we can set them within the *CREATE DATABASE* statement.

For public databases we use the word *SET* followed by the parameters. If you specify more than one, you have to separate them by a comma.

```
admin > CREATE DATABASE publicdb SET block_size=16kB, encryption=RIJNDAEL_256;
```

Private databases set the parameters within the connect string. Here they are preceded by a question mark and separated by an ampersand.

```
no database > co file:///mydbs/privatedb?block_size=16kB&encryption=RIJNDAEL_256
```

These parameters are only evaluated within the first connect. All subsequent connects ignore these parameters, because the database already exists.

All available database parameters are described in detail in the appendix *Database Parameters* of this manual.

2.2. Modify Database

Databases can be modified with the *ALTER DATABASE* statement. For public databases this is similar to the *CREATE DATABASE* statement.

```
admin > ALTER DATABASE publicdb SET BUFFER_SIZE=16MB, BUFFER_CONCURRENCY=4;
```

It is also possible to alter a database directly on a public database.

```
publicdb > ALTER DATABASE SET BUFFER_SIZE=16MB, BUFFER_CONCURRENCY=4;
```

The statement is the same but there is no database name given, because it refers to the current database.

Private databases can also be modified with the ALTER DATABASE statement like public databases.

```
/mydbs/privatedb > ALTER DATABASE SET BUFFER_SIZE=16MB, BUFFER_CONCURRENCY=4;
```

Almost all parameters of the CREATE DATABASE statement can be modified with the ALTER DATABASE statement. An exception is the block size because this would lead to a rebuild of the whole database.

All available database parameters are described in detail in the appendix *Database Parameters* of this manual.

For the complete syntax of the *ALTER DATABASE* statement see *AlterDatabaseStatement*.

2.3. Drop Database

The *DROP DATABASE* statement deletes a public database. It is only available on the admin database. The database is shut down first and then all files are deleted and the database entry in `dblist.ini` is deleted.

```
admin > DROP DATABASE publicdb;
```

if the database has a password set for user tbadmin, IMMEDIATE must be specified.

```
admin > DROP DATABASE publicdb IMMEDIATE;
```

There is no Transbase command for dropping private databases. Instead someone can just delete the entire database directory in the filesystem.

```
# rm -rf /mydbs/privatedb
```

2.4. Register Database

The *REGISTER DATABASE* statement serves to register a database from an existing database directory. This is the preferred way to make a private database available through the Transbase service. Technically the database is inserted into `dblist.ini`.

```
admin > REGISTER DATABASE db FROM DIRECTORY /mydbs/privatedb;
```

Database directories, which are left after command *DEREGISTER DATABASE* ... are well suited for this command.

2.5. Deregister Database

To deregister a public database, use the following statement:

```
admin > DEREGISTER DATABASE db;
```

Only the entry in `dblist.ini` is deleted, but all files of the database remain in the file system. This is the difference to the DROP DATABASE statement. Afterwards the database can be copied to another location and it can be accessed like a private database.

2.6. Boot Database

This statement boots all databases, i.e. recovers them from previous crashes and installs their corresponding shared memories. A database must be booted before it can be accessed by programs.

```
admin > BOOT ALL DATABASES;
```

Use the following statement, if you only want to make one specific database available:

```
admin > BOOT DATABASE publicdb;
```

Note that a database may be booted multiply.

Booting a database may be time consuming when the database had crashed and a lengthy transaction had been active that must be rolled back.

Version conflicts can prevent a database from being booted. In particular, when a database has been created with an old Transbase version, it must be migrated to a new version. See *Migrate Database* below.

If the database cannot be restored from a previous crash, then a disk recovery process must be started manually.

Private databases are booted implicitly during the connect.

2.7. Shutdown Database

This statement shuts all databases down, i.e. saves their buffers persistently to disk and uninstalls their corresponding shared memories and semaphores. After shutdown the database cannot be accessed by programs.

```
admin > SHUTDOWN ALL DATABASES;
```

Use the following statement, if you only want to make one specific database unavailable:

```
admin > SHUTDOWN DATABASE publicdb;
```

If kernel threads are running on the database, an error is returned.

With option IMMEDIATE all active Transbase kernel threads are terminated. As a consequence of this all active transactions on the database are aborted.

```
admin > SHUTDOWN DATABASE publicdb IMMEDIATE;
```

Note that a database may be shut down multiply.

Shut down of a database may be time consuming when the database had crashed and a lengthy transaction had been active that must be rolled back.

All booted databases are also shutdown when the Transbase service is shutdown. And when the service is started again, all databases, which were booted before the last shutdown, are booted automatically.

Private databases are shut down implicitly during the disconnect.

2.8. Migrate Database

Migration is always from the version of the database, to the version of the actual Transbase software. The action is usually very fast, because only changes on the catalog are necessary.

Public databases are migrated with the *MIGRATE DATABASE* statement. There is no statement for migrating all databases.

```
admin > MIGRATE DATABASE publicdb;
```

This procedure is not reversible. Before migration, it is strongly recommended to back up the database using a backup tool such as tbarc (of the original software version).

Private databases are automatically migrated during the first connect with the new Transbase version.

```
no database > connect file:///mydbs/privatedb
```

2.9. Flush Database

The *FLUSH DATABASE* statement is used to force all produced logfile entries in memory to their corresponding logfile. Furthermore all changed blocks in the database buffer pool are written to the diskfiles to speed up the next boot operation.

This statement can be helpful to avoid data loss, if someone expects power failures.

All public databases of a Transbase server can be flushed with the following statement:

```
admin > FLUSH ALL DATABASES;
```

Use the next statement, if you only want to flush one specific database:

```
admin > FLUSH DATABASE publicdb;
```

Private databases are flushed with the following statement:

```
/mydbs/privatedb > FLUSH DATABASE;
```

2.10. Export Database

Numerical values are stored in disk files either in big or little endian format, depending on the CPU of the host system. The *EXPORT DATABASE* statement can be used to make a copy of all diskfiles in the opposite format to provide proper diskfiles for another platform.

With the following statement diskfiles are exported in little endian format to directory */data/export*.

```
admin > EXPORT DATABASE publicdb TO /data/export SET BYTE_ORDER=LITTLE_ENDIAN;
```

Note that this feature is only available for public databases.

2.10.1. Create Database from Export

The creation of a public database from an export is performed with the following statement:

```
admin > CREATE DATABASE publicdb FROM EXPORT /data/export;
```

A private database can be created from an export within the *connect* command:

```
no database > connect file://privatedb?export='/data/export'
```

The diskfiles will not be copied to the database directory and on the other hand if someone drops the database, the diskfiles from the export will not be deleted.

3. Database Information

All relevant configuration parameters of a database can be retrieved from the sysdatabases table on the admin database or from the sysdatabase table on the database in question. These properties match exactly with the database parameters that can be specified in the *CREATE DATABASE* statement. See appendix *The sysdatabase(s) Table* for more information.

The sysdatbases table is only available on the repository database. It contains information of all public databases of the service.

To list all databases, type:

```
admin > SELECT DISTINCT database_name FROM sysdatabases;
```

To find out which databases are available or not, type the following:

```
admin > SELECT database_name, value FROM sysdatabases WHERE property='status';
```

Get information about a specific database:

```
admin > SELECT * FROM sysdatabases WHERE database_name='db';
```

Each private or public database contains the sysdatabase table. It contains information about the current database. Compared to the sysdatabases table, field database_name is missing.

```
publicdb > SELECT * FROM sysdatabase;
```

```
/mydbs/privatedb > SELECT * FROM sysdatabase;
```

4. Database Trace

The trace feature of Transbase is a general mechanism to log *Events* occurring on the database. For each event one line is added to a *Trace-File* which holds information such as time, event, username and also additional event-specific information. Examples for events which can be logged are the occurrence of errors, login attempts, the begin and end of transactions or the execution of DDL-statements. All server tasks working on the same database log into the same database-specific files.

The trace feature has minimal impact on the performance.

Tracing can be switched on or off at creation time or later on using *trace=TRUE|FALSE*.

```
admin > CREATE DATABASE db SET trace=TRUE;
```

```
admin > ALTER DATABASE db SET trace=FALSE;
```

Note that the trace status is remembered during the shutdown-boot phases of the database. Therefore tracing is automatically resumed after boot if it was switched on at shutdown.

All trace settings can be examined by calling

```
admin > SELECT property, value FROM sysdatabases
    WHERE property LIKE 'trace%' AND database_name='db';
```

4.1. Trace Event Categories

All database events, that can be logged are in one of the following categories. Thus the trace facility can be customized according to these categories.

- *APPLIC* logs events transmitted from applications.

  See the *Transbase Call Interface* [tci.xhtml#trace_application] manual to learn how applications can transmit events.

- *CONN* logs any connect, login or disconnect calls to the data-base.

  In case of failed attempts the reasons for the failure (`undefined user`, `wrong passwd` etc.) are included.

- *TA* logs all transaction related events triggered by Begin-, Abort- or Commit-Transaction calls to the database. For distributed transactions, an internal Prepare-To-Commit call is logged too.

- *SQL* logs the execution of SQL statements.

- *STORE* logs the preparation of stored statements.

- *STAT* logs statistics at the end of statements or at the end of a session.

- *ERROR* logs error events that occur when processing requests.

  Error events are roughly distinguished as `Hard` if the transaction is aborted due to this error or `Soft` otherwise.

The following category can be used as a shortcut for logging all categories.

- *ALL* switches all events on.

The set of events to be traced can be specified or changed via the database parameter `trace_events`.

```
admin > ALTER DATABASE db SET trace_events=CONN:TA:ERROR;
```

4.2. The Trace File

By default the trace file of a database has the name *trace* and is located in the database subdirectory *trace*. Name and location of this file can be choosen at creation time or later on using the database option *trace_path*. Transbase will add an extension to the trace file automatically.

The trace file can be changed via

```
admin > ALTER DATABASE db SET trace_path='/data/db/trace';
```

This has the effect that the old file is closed and a new, empty file will be created if tracing is switched on.

Note that the old trace file is not deleted. It can be analyzed or spooled into the database for further evaluation.

The maximum size of one trace file can be specified with

```
admin > ALTER DATABASE db SET trace_file_size=1MB;
```

The maximum number of trace files can be set via

```
admin > ALTER DATABASE db SET trace_file_count=16;
```

If this number is exhausted, the oldest trace file is deleted.

For convenience, the layout of the database trace file is such that it can either be used for being spooled into a database table (spool format) or to be imported to a table calculation program (csv format).

```
admin > ALTER DATABASE db SET trace_syntax=csv;
```

For each event, one line (record) is appended to the trace file. The meaning of the entries of a row are declared by the following schema of a table where the trace file can be spooled to:

```
CREATE TABLE "public"."systrace" WITH IKACCESS
(
    InstanceID      bigint,             -- instance id
    ProcessID       integer,            -- process id
    ClientIP        char(*),            -- IP adress of connected client
    ConnectionID    integer,            -- sequence for session
    Login           char(*),            -- current user name
    Database        char(*),            -- database name, not yet used
    Program         char(*),            -- name of calling program, not yet used
    Seq             bigint,             -- unique identifier
    CurrentTime     datetime[yy:ms],    -- timestamp
    Class           char(*),            -- event class
    Subclass        char(*),            -- event subclass
    TaId            integer,            -- Transaction identifier
    QueryId         integer,            -- Query identifier
    StmtId          integer,            -- Statement identifier
    Query           char(*),            -- SQL statement
    Info2           char(*),            -- more info, depending on class
    Info3           char(*),            -- more info, depending on class
    Info4           char(*),            -- more info, depending on class
    Info5           char(*),            -- more info, depending on class
    Info6           char(*),            -- more info, depending on class
    Info7           char(*)             -- more info, depending on class
)  key is Seq;
```

This table can be filled with the *SPOOL TABLE FROM <file>* statement.

```
db > SPOOL systrace FROM db/trace/trace.log;
```

And the content can be evaluated and filtered with standard SQL statements.

```
db > SELECT * FROM systrace;
```

5. Monitoring

Transbase offers only a rudimental mechanism for database monitoring: the virtual table *sysmonitor*.

```
db > SELECT * FROM sysmonitor;
```

With the statement above information about the Read/Write cache, I/O, transactions, threads and more can be retrieved. The result of the query delivers one record for each client connection. There is one additional record, which contains the sum over all. The values are counters beginning with the start of the session.

The following query delivers for thread 42856 how many blocks were written to disk, this is disk_writecount, and how many blocks were read from bfim files in bfim_readcount.

```
db > SELECT process, disk_wcnt, bfim_rcnt FROM sysmonitor;

  process            disk_wcnt            bfim_rcnt
    42856                   13                    5
```

6. Backup

To protect a database against loss or destruction by disk hardware failure, dump mechanisms are provided.

6.1. Discouraged: Manual Copying of Database Files

In general, a file copy operation on all database files does not produce a fileset which represents a consistent database. If a database is in operation (update transactions running) then a consistent backup can only be produced by a suitable administration command.

6.2. Preconditions and Restrictions

A dump of a database can only be created when the database uses Logging as the recovery method because the dump mechanism is based on the binary log. Dumps are not available for databases which work with before images.

The following is for setting the public database into the state required for dumping databases.

```
admin > ALTER DATABASE db SET recovery_method=LOGGING, log_file_size=100;
```

The former option enables delta logging and the latter option sets the size of each logfile to 100 MB (a value > *1MB* is required).

6.3. Create Dump

A backup of a public database is made with the *DUMP DATABASE* statement.

The following command produces an initial dump into a directory */data/dump_dir*:

```
admin > DUMP DATABASE db TO DIRECTORY /data/dump_dir;
```

Instead of dumping to a directory, a dump into one single file is made by replacing *DIRECTORY* with *FILE* in the dump commands, e.g.

```
admin > DUMP DATABASE db TO FILE /data/dump_file;
```

On UNIX platforms, the dump can also be written onto a sequential drive:

```
admin > DUMP DATABASE db TO FILE /dev/rmt1;
```

When dumping to a stream, i.e. to a file or device, the size of each file to be dumped must not exceed 4GB.

A full dump consists of all diskfiles and of some L*.act logfiles at the time of dumping.

This command creates a directory <dir> and stores all relevant database files into <dir>. Relevant logfiles are stored into a subdirectory D000000.

With this dump, the database can be restored into a state as it was at the time the dump was made.

Any change on the database which is done after the dump has been produced, will of course be lost if the dump is used to replace the database.

To refresh the dump with the most recent changes, one could either make a new Full Dump. A faster method is to complete the dump with the most recent changes as described in the following.

6.4. Update Dump

If there are changes on the database you don't have to make a full dump again. Instead you can make an incremental dump. This is done by the *DUMP DATABASE INCREMENTAL* statement. It is important that the given directory already contains an existing dump of this database. You can update a dump as often as you want.

```
admin > DUMP DATABASE db INCREMENTAL TO DIRECTORY /data/dump_dir;
```

This command adds logfiles of the database to the dump directory thus pushing the dump to the current state of the database.

Performing an Incremental Dump adds one subdirectory Dxxxx (with a running number) to the main directory which then contains all logfiles of the database which were not yet in the dump.

Incremental Dumps may be iterated many times, but restoring a database from a highly accumulated Incremental Dump is a bit slower than restoring from a Full Dump.

6.5. Restore Database

There are two ways of restoring a database from a dump.

If the directory structure of the destroyed database is intact, then in-place recovery is possible. You can use the following statement therefore:

```
admin > ALTER DATABASE db UPDATE FROM DUMP DIRECTORY /data/dump_dir;
```

This command will use the database setting as provided in the configuration file residing in *db*'s database directory. First all diskfiles are overwritten with their older copies from the initial dump. Afterwards the bfims directory is cleaned. Remember that logfiles have been saved in Step1. Finally all logfiles from the initial dump are copied to the bfims directory. The database now is in its initial state as it was when the first dump was made.

The other way of restoring a database is to recover it to another location. A new public database is created with the *CREATE DATABASE* statement.

```
admin > CREATE DATABASE newdb FROM DUMP DIRECTORY /data/dump_dir;
```

And a private database is created within the first connect with an additional DUMP parameter which points to the dump file or directory.

```
no database > connect file://newdb?dump='/data/dump_dir'
```

After the database was recovered it is not booted automatically. The following command makes the database available again:

```
admin > BOOT DATABASE db;
```

7. Replication

Transbase Replication provides functionality for creating copies of databases (replicas), which are frequently updated to ensure consistency between the original and the copies.

Replication can be used for following purposes:

- load balancing: to avoid an overload on one database, requests can be distributed across a set of replicas on different computers. The Transbase service can work as the load balancer when the replicas are organised into *Grids*.
- accessibility: when a computer fails, another computer with a copy of the database is still reachable.
- reliability: if the original database gets corrupted, one of the corresponding replicas can replace the original database.

7.1. Overview

Transbase Replication is based on the client-server architecture.

Every replica (client) communicates with one thread of the corresponding transbase service or the origin. A replica can also act as an origin for other replicas (cascading). Private databases can only be replicas whereas an origin must be a public database.

Replication is based on the "delta logging" transaction recovery method, which writes a sequential binary stream of all database changes into logfiles. The content of these logfiles is also streamed to the replicas, where the logentries are read and applied to the diskfiles.

If a replication service is started for a replica, which actually does not exist, a full dump is transferred from the origin to the replica to create the initial database.

Replicas need not be connected permanently to the origin. They can disconnect at any time, leaving full functional databases. As soon as a replica connects again, all missing logfiles are streamed to the replica and applied.

If there are no changes on the original database, there is also no network traffic to the replicas. But whenever changes are made to the original database, all replicas get the updates immediately, i.e. after each completed transaction.

7.2. Replication Modes

Depending on your needs, Transbase Replication can be run in several modes (asynchronous, semi-synchronous and synchronous).

In asynchronous mode the binary log is transferred to the replicas as soon as possible, but a commiting transaction on the original database does not have to wait until the replicas received the complete log. So in case of a system crash on the origin side, there is no guarantee that all data of already committed transactions has been transferred to the replicas.

In semi-synchronous mode a commiting transaction on the original database must wait until all replicas have received the entire log and written it to disk. Because of old readers on the replicas the changes are possibly not visible at this particular time. But in case of a system crash, the replicas are able to replay the full log.

Synchronous replication works like semi-synchronous replication, but as long as the log has not been replayed on a replica, no new transactions are permitted on this replica.

It is possible to run one replica in synchronous mode and another one in asynchronous mode.

7.3. Configure Origin

Replication is possible between different types of operating systems, e.g. the operating system hosting the original database is Linux and an associated replica runs under Windows. But it is not possible to replicate between two machines with different byte order. Both sides must run either on little endian machines or big endian machines.

All updates/write operations have to be made on the original database. The replicas are read-only databases.

The transbase replication service only works on encrypted databases with delta logging and not with before image logging. In order to do so, set parameter *encryption* to *RIJNDAEL_256*, parameter *recovery_method* must be switched to value *logging* and the size of the logfiles must be set to a value > *100MB*. It is important to choose the log_window big enough, because if a replica is for a long time offline it can happen that the required log files are already deleted on the original database.

```
admin > CREATE DATABASE origin
    SET encryption=RIJNDAEL_256, log_file_size=100, recovery_method=logging;
```

Because there is only one binary delta log for an entire database, it is not possible to replicate selected dataspaces or tables. Only the entire database can be replicated.

7.4. Create Replica

A replica is created by the following command:

```
admin > CREATE DATABASE replica FROM REPLICATION //www.transaction.de:2024/origin;
```

The given origin must at least contain a database name. But it can also be a full specified connection string (with protocol, host and port).

With option *dump* the replica is rebuilt from a dump of the original database. The advantage is, that the dump doesn't have to be transferred over the network. Only the latest changes have to be transferred over the network, but not the big part of the database.

```
admin > CREATE DATABASE replica
    FROM REPLICATION //www.transaction.de:2024/origin SET dump=/data/dump;
```

A private database can also be created from replication. Therefore you must specify the location of the original database with the parameter *replication_origin*.

```
no database > connect file://replica?replication_origin='//www.transaction.de:2024/origin'
```

Note that the replica is not booted afterwards.

7.5. Update Replica

With the following command a replica can be updated to the current state of the original database. Afterwards the service is terminated.

```
admin > ALTER DATABASE replica
    UPDATE FROM REPLICATION ONCE;

/home/databases/replica > ALTER DATABASE
    UPDATE FROM REPLICATION ONCE;
```

During the update operation the database property *replication_mode* switches from *none* to *once*. After the operation has finished it switches back to *NONE*.

To start the replication of a replica as a continiuous service, type:

```
admin > ALTER DATABASE replica
    UPDATE FROM REPLICATION START [SYNCHRONOUS|ASYNCHRONOUS|SEMISYNCHRONOUS];

/home/databases/replica > ALTER DATABASE
    UPDATE FROM REPLICATION START [SYNCHRONOUS|ASYNCHRONOUS|SEMISYNCHRONOUS];
```

The options *SYNCHRONOUS*, *ASYNCHRONOUS* and *SEMISYNCHRONOUS* specify, whether the replica is updated synchronously, semi-synchronously or asynchronously (see *Replication Modes*). If omitted, it defaults to *ASYNCHRONOUS*. The database property *replication_mode* changes to this value.

Note that this service does not halt, if the replica got the latest updates from the origin or the origin is no longer reachable.

With function *replication_status()*, the status (*active* or *inactive*) of the replication service can be checked:

```
admin > SELECT replication_status()@replica;

/home/databases/replica > SELECT replication_status();
```

If the function returns *active*, then the replica is currently connected to the original database. If the function returns *inactive* and the value of database property *replication_mode* equals *synchronous*, *asynchronous* or *semisynchronous*, then there is no active connection between the replica and the original database at the moment, but the replica tries to connect every 30 seconds.

To stop the replication service of a replica, type:

```
admin > ALTER DATABASE replica UPDATE FROM REPLICATION STOP;

/home/databases/replica > ALTER DATABASE UPDATE FROM REPLICATION STOP;
```

The database property *replication_mode* changes after this operation to *none*.

The date of the last update on the replica is retrieved by the *last_update()* function.

```
admin > SELECT last_update()@replica;

/home/databases/replica > SELECT last_update();
```

7.6. Convert Replica to Read/Write Database

A replica can be turned into a read/write database if the value of database property *replication* is set to *FALSE*.

```
admin > ALTER DATABASE replica SET replication=FALSE;
```

```
/home/databases/replica > ALTER DATABASE SET replication=FALSE;
```

7.7. Grid Administration

This chapter describes how to administrate grids.

A Transbase grid consists of several public databases. These databases may be controlled by different databases servers located on different hosts or systems.

Grids are recursive, which means that a grid can also be a member of another grid.

When an application wants to connect to a grid, it is redirected to one of the grid's databases. So grids are best used for load balancing to avoid an overload on one database server. The task of the load balancer is performed by the Transbase server.

Like public databases, grids can only be administrated by the Transbase Service.

7.7.1. Create Grid

A database grid is created by the following statement:

```
admin > CREATE GRID grid WITH db1, db2, ... ;
```

Currently a maximum number of 11 database entries are allowed for one grid.

7.7.2. Modify Grid

Databases can be added or removed to/from a grid by the ALTER GRID statement.

To add database db3 to grid grid, type the following:

```
admin > ALTER GRID grid ADD DATABASE db3;
```

To remove database db1, use this statement:

```
admin > ALTER GRID grid DROP DATABASE db1;
```

Only the entry in the grid is removed. The database itself remains unaltered.

7.7.3. Drop Grid

A database grid is deleted by the following statement:

```
admin > DROP GRID grid;
```

Only the grid is removed. All databases remain unaltered.

7.7.4. Grid Information

The sysdatabases table contains information about all grids.

To list all grids type the following:

```
admin > SELECT DISTINCT database_name FROM sysdatabases
    WHERE property='participant';
```

Get all database entries of one specific grid:

```
admin > SELECT value FROM sysdatabases
    WHERE property='participant'AND database_name='grid';
```

8. Publication

This section gives an introduction into the publishing process. For a deeper insight take a look at the *Transbase Publishing Guide* [tbcd.xhtml#Introduction].

8.1. Preconditions

A public database serves as the starting point for the publishing process.

```
admin > CREATE DATABASE dbstd;
```

This command creates a logically empty database *dbstd*. In the sequel, the database might be filled with user data via SQL commands.

8.2. Create Publication

With the following statement romfiles are created in the given directory */data/publication*.

```
admin > PUBLISH DATABASE dbstd TO /data/publication;
```

8.3. Create Retrieval Database

The creation of a retrieval database to work upon the romfiles data is performed with command:

```
admin > CREATE DATABASE dbretr FROM PUBLICATION /data/publication;
```

```
no database> connect file://dbretr?publication='/data/publication'
```

All data on a retrieval database also can be updated. The romfiles, of course, remain unchanged and all changes and new data accumulate on the shadow files.

8.4. Incremental Publishing

To publish only the updates made on the retrieval database, use the following statement:

```
admin > PUBLISH DATABASE dbretr INCREMENTAL TO /data/publication;
```

The original romfiles in the publication directory remain unchanged, but all database updates are stored in additional files in a new subdirectory.

It is not possible to apply the newly created files directly to a retrieval database. Instead the existing database must be dropped and recreated from the updated publication directory.

```
admin > DROP DATABASE dbretr;
```

```
admin > CREATE DATABASE dbretr FROM PUBLICATION /data/publication;
```

The whole procedure can be repeated to publish prospective updates.

9. Crowd Service

This chapter explains the basic tasks of the crowd service. For a deeper insight take a look at the *Transbase Crowd Queries Guide* [crowd.xhtml#Introduction].

9.1. Configure Crowd Service

Databases specify the crowd master, to whom they connect to, through the database property *crowd_master*:

```
admin > ALTER DATABASE db
    SET crowd_master='//localhost:2024/crowd_master?crowd=my_crowd';

/home/databases/db > ALTER DATABASE
    SET crowd_master='//localhost:2024/crowd_master?crowd=my_crowd';
```

The value must be a connection string to a database on the network.

9.2. Start Crowd Service

With the following command a database connects to its crowd master, which is configured by the database property *crowd_master*:

```
admin > ALTER DATABASE db SET crowd_connect=TRUE;

/home/databases/db > ALTER DATABASE SET crowd_connect=TRUE;
```

Note that this service does not halt, if the crowd master is no longer reachable.

With function *crowd_status()*, the status (*active* or *inactive*) of the crowd service can be checked:

```
admin > SELECT crowd_status()@db;

/home/databases/db > SELECT crowd_status();
```

If the function returns *active*, then the database is currently connected to the crowd master. If the function returns *inactive*, then there is no active connection between the database and the crowd master at the moment, but the database tries to connect every 30 seconds until the database property *crowd_connect* is set to *FALSE* or the database is shutdown. If the database is booted again, the service is started automatically, if the database property *crowd_connect* has the value *TRUE*.

9.3. Stop Crowd Service

The crowd service of a database can be stopped through changing the value of database property *crowd_connect* to *FALSE*.

```
admin > ALTER DATABASE db SET crowd_connect=FALSE;

/home/databases/db > ALTER DATABASE SET crowd_connect=FALSE;
```

Appendix A. Administration Language

TB/SQL provides several administrative operations for databases and grids.

A.1. Overview of AdministrationStatement

Syntax:

```
[1]   AdministrationStatement ::= CreateDatabaseStatement |
                                  AlterDatabaseStatement |
                                  RegisterDatabaseStatement |
                                  DeregisterDatabaseStatement |
                                  BootDatabaseStatement |
                                  ShutdownDatabaseStatement |
                                  MigrateDatabaseStatement |
                                  FlushDatabaseStatement |
                                  DropDatabaseStatement |
                                  PublishDatabaseStatement |
                                  ExportDatabaseStatement |
                                  DumpDatabaseStatement |
                                  CreateGridStatement |
                                  AlterGridStatement |
                                  DropGridStatement
```

A.2. Literals

This section list all basic literals which are used in the administration language. A detailed description can be found in the *TB/SQL-Manual* [sql.xhtml#sql_literal].

```
[2]          BoolLiteral ::= FALSE | TRUE
[3]       IntegerLiteral ::= 0 | {1..9}[{0..9}]...
[4]       NumericLiteral ::= IntegerLiteral . [ IntegerLiteral ] | . IntegerLiteral
[5]        StringLiteral ::= CharacterLiteral | UnicodeLiteral | USER
[6]     CharacterLiteral ::= <sequence of characters enclosed in single quotes>
[7]      UnicodeLiteral ::= <0u followed by sequence of hexadecimal characters>
[8]     DirectoryLiteral ::= FileLiteral
[9]          FileLiteral ::= StringLiteral | <any sequence of alphanumeric characters>
```

A.3. Database Parameters

This section describes all parameters for creating and modifying databases.

Syntax:

```
[10]              DBParamBlockSize : := BLOCK_SIZE = { 1 | 2 | 4 | 8 | 16 | 32 }
[11]       DBParamBufferConcurrency : := BUFFER_CONCURRENCY = IntegerLiteral
[12]             DBParamBufferSize : := BUFFER_SIZE = StringLiteral
[13]          DBParamCaseInsensitive : := CASE_INSENSITIVE = BoolLiteral
[14]       DBParamCommCompression : := COMMUNICATION_COMPRESSION = BoolLiteral
[15]         DBParamConnectionLimit : := CONNECTION_LIMIT = IntegerLiteral
[16]                DBParamSize : := SIZE = StringLiteral
[17]              DBParamDBConf : := DBCONF = StringLiteral
[18]                DBParamDump : := DUMP = StringLiteral
[19]          DBParamEncryption : := ENCRYPTION = { NONE | RIJNDAEL_256 | AES_256_CBC }
[20]         DBParamLogFileSize : := LOG_FILE_SIZE = StringLiteral
[21]        DBParamLogBlockSize : := LOG_BLOCK_SIZE = IntegerLiteral
[22]          DBParamLogWindow : := LOG_WINDOW = { UNLIMITED | IntegerLiteral }
[23]         DBParamLogWriteThru : := LOG_WRITETHRU = { COMMIT | DEMAND }
[24]              DBParamName : := NAME = StringLiteral
[25]          DBParamPassword : := PASSWORD = StringLiteral
[26]              DBParamPath : := PATH = StringLiteral
[27]       DBParamPublication : := PUBLICATION = StringLiteral
[28]      DBParamQueryThreading : := QUERY_THREADING = { OFF | MEDIUM | HIGH }
[29]      DBParamRecoveryMethod : := RECOVERY_METHOD = { LOGGING | BEFOREIMAGE }
[30]        DBParamRecoveryPath : := RECOVERY_PATH = StringLiteral
[31]        DBParamSchemaDefault : := SCHEMA_DEFAULT = { PUBLIC | USER }
[32]        DBParamSortBufferSize : := SORT_BUFFER_SIZE = StringLiteral
[33]           DBParamTempPath : := TEMP_PATH = StringLiteral
[34]           DBParamByteOrder : := BYTE_ORDER = { LITTLE_ENDIAN | BIG_ENDIAN }
[35]              DBParamExport : := EXPORT = StringLiteral

[36] DBParamMyriadSecurityClasses : := MYRIAD_SECURITY_CLASSES = MyriadSecurityClass [ Myr-
                                       iadSecurityClassDelimiter MyriadSecurityClass ]
[37]          MyriadSecurityClass : := MyriadSecurityClassLeftBrace MyriadSecurityClassNumber Myr-
                                       iadSecurityClassDelimiter MyriadSecurityClassRate MyriadSecu-
                                       rityClassRightBrace
[38]    MyriadSecurityClassNumber : := IntegerLiteral
[39]      MyriadSecurityClassRate : := NumericLiteral
[40]  MyriadSecurityClassDelimiter : := :
[41]  MyriadSecurityClassLeftBrace : := {
[42] MyriadSecurityClassRightBrace : := }

[43]      DBParamReplicationOrigin : := REPLICATION_ORIGIN = StringLiteral
[44]      DBParamReplicationMode : := REPLICATION_MODE = { NONE | ONCE | ASYNCHRONOUS
                                       | SYNCHRONOUS | SEMISYNCHRONOUS }
[45]          DBParamReplication : := REPLICATION = BoolLiteral

[46]         DBParamCrowdMaster : := CROWD_MASTER = StringLiteral
[47]         DBParamCrowdConnect : := CROWD_CONNECT = BoolLiteral

[48]               DBParamTrace : := TRACE = BoolLiteral
[49]                 TraceEvent : := ERROR | CONN | TA | SQL | STORE | STAT
[50]              TraceEventList : := TraceEvent [ TraceEventDelimiter TraceEvent ]...
[51]          TraceEventDelimiter : := :
[52]          DBParamTraceEvents : := TRACE_EVENTS = { ALL | TraceEventList }
[53]       DBParamTraceFileCount : := TRACE_FILE_COUNT = IntegerLiteral
[54]       DBParamTraceFileSize : := TRACE_FILE_SIZE = StringLiteral
[55]         DBParamTracePath : := TRACE_PATH = StringLiteral
[56]         DBParamTraceSyntax : := TRACE_SYNTAX = { CSV | SPOOL }
```

Explanation:

- DBParamBlockSize sets the page size in KB for the database. If omitted, page size defaults to 8k. See Tuning Guide.
- DBParamBufferConcurrency specifies the number of shared memory areas allocated for the database
- DBParamBufferSize specifies the size of a single shared memory area.
- DBParamCaseInsensitive : if TRUE, the database will be case-insensitive, i.e. all identifiers will be mapped to upper-case. If omitted, case-insensitivity defaults to off.
- DBParamCommCompression enables or disables communication compression between client and server. The default is `FALSE` .
- DBParamConnectionLimit sets the maximum number of concurrent sessions on the database. It cannot be set higher than permitted by the Transbase server license.
- DBParamSize sets the size of the initial datafile. e.g. size=1024MB
- DBParamDBConf create a database with settings identical to those referenced by the dbconf files.
- DBParamDump specifies the location of the dump file or directory.
- DBParamEncryption enables database encryption. The default is `NONE` . Encryption is a creation parameter and cannot be changed later. `AES_256_CBC` supports hardware acceleration, therefore it should be preferred over `RIJNDAEL_256` .
- DBParamLogFileSize specifies the size of logfiles in MBytes.
- DBParamLogBlockSize specifies the unit of I/O between logfiles and logbuffer cache in KB. According to the external medium used, it might be useful to adapt it to the physical I/O unit. The default size is the size of the database blocks.
- DBParamLogWindow : if disk recovery is switched on for the database, the expiration of the logfiles can be set via this option. The logfiles are removed after the chosen amount of days, if they are no longer needed for transaction recovery. If omitted, expiration defaults to unlimited.
- DBParamLogWriteThru specifies how often the transaction log is flushed to disk. Since log flush may be a lengthy operation, this parameter has significant impact on performance for update transactions.
  - `COMMIT` means that the log is flushed upon commit of each update transaction. Thus committed transactions are always guaranteed to be persistent, no committed transaction will be lost due to a machine crash. Additionally Transbase will perform a log flush on demand in order to guarantee the consistency of disk files. A log flush on demand will happen very seldom for large data caches and for short transactions. It will happen more often when long transactions are processed or when data cache is small.
  - The default value is `DEMAND` which means that the log is not flushed upon commit of each transaction but on demand to avoid data corruption. This setting means that transactions may be lost if committed shortly before a crash. Performance for update transactions may be improved significantly.
- DBParamName sets the name of the database.
- DBParamPassword specifies the tbadmin password. If omitted, the tbadmin password is the empty string.
- DBParamPath sets home directory of the database. If omitted the database home directory will be located in the current directory and named *$TRANSBASE/databases/<DatabaseName>* .
- DBParamPublication specifies the location of the romfiles.
- DBParamQueryThreading configures the multithreading behaviour of a single query being processed.
  - `HIGH` 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.

  - `MEDIUM` uses 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.
  - `OFF` means there is no query-internal parallel processing at all. This is the default.
- DBParamRecoveryMethod changes the recovery method of the database. If set to *BEFOREIMAGE* , database is switched to Before-Image-Logging. If set to *LOGGING* , database is switched to Delta-Logging.
- DBParamRecoveryPath must be a valid path name for the Before Image Disk Recovery directory. If omitted, the pathname defaults to bfim/ in the database home directory.
- DBParamSchemaDefault specifies the default schema of database objects, if they are created without specification of a schema. If omitted, schema defaults to PUBLIC.
- DBParamSortBufferSize sets the size (in kB) of the local (sorter) cache which is allocated for each database instance. If omitted, the local cache size defaults to 2 MB. See Tuning Guide.
- DBParamTempPath must be a valid path name for the scratch directory. If omitted, the pathname defaults to scratch/ in the database home directory.
- DBParamByteOrder sets the byte order of the disk files. If omitted, defaults to the byte order of the host.
- DBParamExport specifies the location of the diskfiles.
- DBParamMyriadSecurityClasses specifies the security classes of a myriad archive. This is a colon-separated list containing for each security class it's security number and the corresponding percent rate.
- DBParamReplicationOrigin specifies the connection string to the original database.
- DBParamReplicationMode sets the replication mode for updating a replica If set to *ONCE* , the replica will be updated to the current state of the original database. Afterwards the service is terminated. With *ASYNCHRONOUS* , *SYNCHRONOUS* and *SEMISYNCHRONOUS* the replication is started as a continuous service in the background. If set to *NONE* , the service will be stopped.
- With DBParamReplication a replica can be altered to a read/write standard database.
- DBParamCrowdMaster specifies the connection string for the crowd service.
- DBParamCrowdConnect enables or disables the crowd service.
- DBParamTrace switches database tracing on or off. The list of events to be recorded is preserved when it is switched off, so that they can be reactivated in the configuration chosen.
- DBParamTraceFileCount sets the maximum number of trace files. If this number is exhausted, the oldest trace file is deleted.
- DBParamTracePath must be a valid path name for the trace files.
- DBParamTraceFileSize sets the maximum size of each trace file.
- DBParamTraceSyntax sets the format of the trace files. *spool* means format for Transbase spooler, *csv* means comma separated values (e.g. for Microsoft EXCEL et.al.)
- DBParamTraceEvents sets the events to be logged into the trace file.

A.4. CreateDatabaseStatement

Serves to create a database, optionally from dump, replication or publication.

Syntax:

```
[57]                DatabaseName ::= DirectoryLiteral
[58]                  DeviceName ::= DirectoryLiteral
[59]       CreateDatabaseStatement ::= CREATE DATABASE DatabaseName CreateDatabaseSpec
[60]           CreateDatabaseSpec ::= CreateDatabaseStandardSpec |
                                      CreateDatabaseDumpSpec |
                                      CreateDatabaseReplicationSpec |
                                      CreateDatabasePublicationSpec |
                                      CreateDatabaseExportSpec
[61]   CreateDatabaseStandardSpec ::= [ SET CreateDatabaseStandardParam [, CreateDatabaseStandard-
                                      Param ]... ]
[62]  CreateDatabaseStandardParam ::= DBParamBlockSize |
                                      DBParamBufferConcurrency |
                                      DBParamBufferSize |
                                      DBParamCaseInsensitive |
                                      DBParamCommCompression |
                                      DBParamConnectionLimit |
                                      DBParamSize |
                                      DBParamDBConf |
                                      DBParamEncryption |
                                      DBParamLogFileSize |
                                      DBParamLogBlockSize |
                                      DBParamLogWindow |
                                      DBParamLogWriteThru |
                                      DBParamPassword |
                                      DBParamPath |
                                      DBParamQueryThreading |
                                      DBParamRecoveryMethod |
                                      DBParamRecoveryPath |
                                      DBParamSchemaDefault |
                                      DBParamSortBufferSize |
                                      DBParamTempPath |
                                      DBParamCrowdMaster |
                                      DBParamTrace |
                                      DBParamTraceEvents |
                                      DBParamTraceFileCount |
                                      DBParamTraceFileSize |
                                      DBParamTracePath |
                                      DBParamTraceSyntax |
                                      DBParamMyriadSecurityClasses
[63]       CreateDatabaseDumpSpec ::= FROM DUMP DumpSpec |
                                      SET DBParamDump
[64]                    DumpSpec ::= FILE { FileLiteral | DeviceName } | DIRECTORY DirectoryLiteral
[65] CreateDatabaseReplicationSpec ::= FROM REPLICATION DatabaseName [ SET CreateData-
                                      baseReplicationParam [, CreateDatabaseReplicationParam ] ... ] |
                                      SET DBParamReplicationOrigin [, CreateDatabaseReplication-
                                      Param ] ... }
[66] CreateDatabaseReplicationParam ::= DBParamDump | DBParamPath
[67]  CreateDatabasePublicationSpec ::= FROM PUBLICATION [DIRECTORY] DirectoryName [ SET
                                      CreateDatabasePublicationParam [, CreateDatabasePublication-
                                      Param ] ... ] | SET DBParamPublication [, CreateDatabasePublica-
                                      tionParam ]...
[68] CreateDatabasePublicationParam ::= DBParamBufferConcurrency |
                                      DBParamBufferSize |
                                      DBParamCommCompression |
                                      DBParamConnectionLimit |
                                      DBParamLogFileSize |
                                      DBParamLogWriteThru |
                                      DBParamPath |
```

DBParamQueryThreading |
DBParamRecoveryPath |
DBParamSchemaDefault |
DBParamSortBufferSize |
DBParamTempPath |
DBParamTrace |
DBParamTraceEvents |
DBParamTraceFileCount |
DBParamTracePath |
DBParamTraceSyntax

[69] CreateDatabaseExportSpec ::= FROM EXPORT [DIRECTORY] DirectoryName [SET CreateDatabaseExportParam [, CreateDatabaseExportParam] ...] | SET DBParamExport [, CreateDatabaseExportParam]...

[70] CreateDatabaseExportParam ::= DBParamBufferConcurrency |
DBParamBufferSize |
DBParamCommCompression |
DBParamConnectionLimit |
DBParamLogFileSize |
DBParamLogWriteThru |
DBParamPath |
DBParamQueryThreading |
DBParamRecoveryPath |
DBParamSchemaDefault |
DBParamSortBufferSize |
DBParamTempPath |
DBParamTrace |
DBParamTraceEvents |
DBParamTraceFileCount |
DBParamTracePath |
DBParamTraceSyntax

Explanation:

- `CREATE DATABASE DatabaseName SET ...` creates a standard database.
- `CREATE DATABASE DatabaseName SET MYRIAD_SECURITY_CLASSES=...` creates a myriad archive.
- `CREATE DATABASE DatabaseName FROM DUMP ...` creates a database from a dump. Alternatively, the dump can be specified by *SET DUMP ...* .
- `CREATE DATABASE DatabaseName FROM REPLICATION ...` creates a replica. Alternatively, the original database can be specified by *SET REPLICATION_ORIGIN ...* .
- `CREATE DATABASE DatabaseName FROM PUBLICATION ...` creates a database from a publication. Alternatively, the publication directory can be specified by *SET PUBLICATION ...* .
- `CREATE DATABASE DatabaseName FROM EXPORT ...` creates a database from an export. Alternatively, the export directory can be specified by *SET EXPORT ...* .

Note

- Databases created from dump, export, replication and publication are not booted afterwards.
- Parameters *DBParamPath* , *DBParamTempPath* and *DBParamRecoveryPath* are not available for private databases.

```
CREATE DATABASE stddb
CREATE DATABASE stddb SET CONNECTION_LIMIT=100, BLOCK_SIZE=32

CREATE DATABASE myriadarchive SET MYRIAD_SECURITY_CLASSES={1:30}:{2:20}:{3:50}

CREATE DATABASE publdb FROM PUBLICATION DIRECTORY '/data/publ'
  SET QUERY_THREADING=HIGH
CREATE DATABASE publdb SET PUBLICATION='/data/publ', QUERY_THREADING=HIGH

CREATE DATABASE dumpdb FROM DUMP DIRECTORY '/data/dump_dir'
CREATE DATABASE dumpdb SET DUMP='/data/dump_dir'

CREATE DATABASE replica FROM REPLICATION origin SET DUMP='/data/dump_file'
CREATE DATABASE replica SET REPLICATION_ORIGIN=origin, DUMP='/data/dump_file'

CREATE DATABASE expdb FROM EXPORT DIRECTORY '/data/export'
CREATE DATABASE expdb SET EXPORT='/data/export'
```

A.5. AlterDatabaseStatement

Serves to alter a database or to update a database from dump or replication.

Syntax:

[71] AlterDatabaseStatement ::= ALTER DATABASE DatabaseName AlterDatabaseSpec

[72] AlterDatabaseSpec ::= AlterDatabaseStandardSpec |
AlterDatabaseDumpSpec |
AlterDatabaseReplicationSpec

[73] AlterDatabaseStandardSpec ::= SET AlterDatabaseStandardParam [, AlterDatabaseStandardParam]...

[74] AlterDatabaseStandardParam ::= DBParamBufferConcurrency |
DBParamBufferSize |
DBParamCaseInsensitive |
DBParamCommCompression |
DBParamConnectionLimit |
DBParamLogFileSize |
DBParamLogWindow |
DBParamLogWriteThru |
DBParamName |
DBParamPassword |
DBParamQueryThreading |
DBParamRecoveryMethod |
DBParamRecoveryPath |
DBParamReplicationOrigin |
DBParamReplication |
DBParamCrowdMaster |
DBParamCrowdConnect |
DBParamSortBufferSize |
DBParamTempPath |
DBParamTrace |
DBParamTraceEvents |
DBParamTraceFileCount |
DBParamTraceFileSize |
DBParamTracePath |
DBParamTraceSyntax

[75] AlterDatabaseDumpSpec ::= UPDATE FROM DUMP DumpSpec |
SET DBParamDump

[64] DumpSpec ::= FILE { FileLiteral | DeviceName } | DIRECTORY DirectoryLiteral

[76] AlterDatabaseReplicationSpec ::= UPDATE FROM REPLICATION ReplicationUpdateMode | SET DBParamReplicationMode
[77] ReplicationUpdateMode ::= ONCE | START [SYNCHRONOUS | ASYNCHRONOUS | SEMISYNCHRONOUS] | STOP

Explanation:

- *`ALTER DATABASE DatabaseName SET ...`* alters a standard database.
- *`ALTER DATABASE DatabaseName UPDATE FROM DUMP ...`* updates a database from a dump. Alternatively, the dump can be specified by *ALTER DATBASE DatabaseName SET DUMP ...* .
- *`ALTER DATABASE DatabaseName UPDATE FROM REPLICATION ...`* updates a replica. Alternatively, the following syntax can be used: *ALTER DATABASE DatabaseName SET REPLICATION_MODE ...* . *ONCE* updates a replica to the current state of the original database. Afterwards the service is terminated. *SYNCHRONOUS* | *ASYNCHRONOUS* | *SEMISYNCHRONOUS* starts the replication of a replica as a continuous service. *STOP* | *NONE* stops the continuous service.

```
ALTER DATABASE stddb SET CONNECTION_LIMIT=100, QUERY_THREADING=HIGH

ALTER DATABASE dumpdb UPDATE FROM DUMP DIRECTORY '/data/dump_dir'
ALTER DATABASE dumpdb SET DUMP='/data/dump_dir'

ALTER DATABASE replica UPDATE FROM REPLICATION ONCE
ALTER DATABASE replica SET REPLICATION_MODE=ONCE

ALTER DATABASE replica UPDATE FROM REPLICATION START SYNCHRONOUS
ALTER DATABASE replica SET REPLICATION_MODE=SYNCHRONOUS

ALTER DATABASE replica UPDATE FROM REPLICATION STOP
ALTER DATABASE replica SET REPLICATION_MODE=NONE

ALTER DATABASE db SET CROWD_MASTER='//localhost:2024/crowd_master'
ALTER DATABASE db SET CROWD_CONNECT=TRUE
```

A.6. RegisterDatabaseStatement

Serves to register a database from an existing database directory.

Syntax:

[78] RegisterDatabaseStatement ::= REGISTER DATABASE DatabaseName RegisterDatabaseSpec
[79] RegisterDatabaseSpec ::= FROM [DIRECTORY] DirectoryLiteral [SET RegisterDatabaseParam [, RegisterDatabaseParam]...]
[80] RegisterDatabaseParam ::= DBParamBufferConcurrency | DBParamBufferSize | DBParamCommCompression | DBParamConnectionLimit | DBParamLogFileSize | DBParamLogWriteThru | DBParamQueryThreading | DBParamSortBufferSize | DBParamTrace | DBParamTraceEvents | DBParamTraceFileCount | DBParamTraceSyntax

Explanation:

- *`REGISTER DATABASE DatabaseName ...`* registers a database from an existing database directory into dblist.ini.

```
REGISTER DATABASE db FROM /databases/db SET CONNECTION_LIMIT=100
```

A.7. DeregisterDatabaseStatement

Serves to deregister a database.

Syntax:
[81] DeregisterDatabaseStatement ::= DEREGISTER DATABASE DatabaseName

Explanation:

- *`DEREGISTER DATABASE DatabaseName ...`* deletes the entry of the database in dblist.ini, but all files of the database remain in the file system.

```
DEREGISTER DATABASE db
```

A.8. BootDatabaseStatement

This statement boots databases, i.e. recovers them from previous crashes and installs their corresponding shared memories. A database must be booted before it can be accessed by programs.

Syntax:
[82] BootDatabaseStatement ::= BOOT DATABASE DatabaseName |
BOOT ALL [DATABASES]

Explanation:

- *`BOOT DATABASE DatabaseName`* boots database DatabaseName.
- *`BOOT ALL [ DATABASES ]`* boots all databases on the local machine.

```
BOOT DATABASE db
BOOT ALL DATABASES
```

A.9. ShutdownDatabaseStatement

This statement shuts databases down, i.e. saves their buffers persistently to disk and uninstalls their corresponding shared memories and semaphores. After shutdown the database cannot be accessed by programs.

Syntax:
[83] ShutdownDatabaseStatement ::= SHUTDOWN DATABASE DatabaseName [IMMEDIATE] |
SHUTDOWN ALL [DATABASES] [IMMEDIATE]

Explanation:

- *SHUTDOWN DATABASE DatabaseName* database DatabaseName shuts down.
- *SHUTDOWN ALL [DATABASES]* all databases on the local machine are shut down.
- *[IMMEDIATE]* terminates all active database connections to databases operated by this service thereby aborting any active transactions on the databases.

```
SHUTDOWN DATABASE db IMMEDIATE
SHUTDOWN ALL DATABASES
```

A.10. MigrateDatabaseStatement

This statement is used to migrate a database that has been created with an older Transbase version to the current version.

Syntax:
[84] MigrateDatabaseStatement ::= MIGRATE DATABASE DatabaseName

Explanation:

- *MIGRATE DATABASE DatabaseName* migrates database DatabaseName.

```
MIGRATE DATABASE db
```

A.11. FlushDatabaseStatement

This statement is used to force all produced logfile entries in memory to their corresponding logfile. Furthermore all changed blocks in the database buffer pool are written to the diskfiles to speed up the next boot operation.

Syntax:
[85] FlushDatabaseStatement ::= FLUSH DATABASE DatabaseName |
FLUSH ALL [DATABASES]

Explanation:

- *FLUSH DATABASE DatabaseName* flushes database DatabaseName.
- *FLUSH ALL [DATABASES]* flushes all databases on the local machine.

```
FLUSH DATABASE db
FLUSH ALL DATABASES
```

A.12. DropDatabaseStatement

This statement deletes a database. The database is shut down first and then all files are deleted and the database entry in dblist.ini is deleted.

Syntax:
[86] DropDatabaseStatement ::= DROP DATABASE DatabaseName [IMMEDIATE]

Explanation:

- *`DROP DATABASE DatabaseName`* if database DatabaseName has no password set for user tbadmin, the database will be dropped.
- *`[ IMMEDIATE ]`* drops a database even if the database has a password set for user tbadmin.

```
DROP DATABASE db
```

A.13. PublishDatabaseStatement

This statement is used for creating romfiles. See the *Transbase Publishing Guide* [tbcd.xhtml] for more information.

Syntax:
[87] PublishDatabaseStatement ::= PUBLISH DATABASE DatabaseName [INCREMENTAL [XOR]]
TO [DIRECTORY] DirectoryLiteral

Explanation:

- *`PUBLISH DATABASE DatabaseName`* creates romfiles and writes ("flushes") the contents of the diskfile(s) to the romfiles.
- *`PUBLISH DATABASE DatabaseName INCREMENTAL`* updates a romfiles directory with new data from the database.
- *`[XOR]`* specifies whether or not XOR technique should be used for calculating the difference.
- *`TO [DIRECTORY] DirectoryName`* specifies the output directory for the romfiles.

```
PUBLISH DATABASE db TO DIRECTORY /publish_directory/db_pub
PUBLISH DATABASE db INCREMENTAL TO /publish_directory/db_pub
```

A.14. ExportDatabaseStatement

This statement is used to export diskfiles either in big or little endian format.

Syntax:
[88] ExportDatabaseStatement ::= EXPORT DATABASE DatabaseName
TO [DIRECTORY] DirectoryLiteral
[SET DBParamByteOrder]

Explanation:

- *`EXPORT DATABASE DatabaseName TO [DIRECTORY] DirectoryName`* writes all diskfiles of the database to the given directory.

- *SET BYTE_ORDER=LITTLE_ENDIAN* converts all diskfiles to little endian format, if the current host is big endian. Otherwise it has no effect.

```
EXPORT DATABASE db TO DIRECTORY /export_directory/db_exp SET BYTE_ORDER=BIG_ENDIAN
```

A.15. DumpDatabaseStatement

This statement is used for creating database dumps.

Syntax:

[89] DumpDatabaseStatement ::= DUMP DATABASE DatabaseName [INCREMENTAL] TO DumpSpec

[64] DumpSpec ::= FILE { FileLiteral | DeviceName } | DIRECTORY DirectoryLiteral

Explanation:

- *DUMP DATABASE DatabaseName TO ...* creates a full dump of database DatabaseName.
- *[INCREMENTAL]* only logfile changes since the last (differential) dump are appended to the given dump.
- *TO FILE FileName | DeviceName* specifies the output file or device for the dump.
- *TO DIRECTORY DirectoryName* specifies the output directory for the dump.

```
DUMP DATABASE db TO DIRECTORY /dump_directory/db_dump
DUMP DATABASE db INCREMENTAL TO DIRECTORY /dump_directory/db_dump
```

A.16. CreateGridStatement

This statement creates a new database grid on the local machine.

Syntax:

[90] CreateGridStatement ::= CREATE GRID GridName WITH DatabaseName [, DatabaseName]...

[91] GridName ::= DatabaseName

Explanation:

- *CREATE GRID GridName WITH DatabaseName* creates a grid GridName containing database DatabaseName

```
CREATE GRID grid1 WITH db1, db2, db3
```

A.17. AlterGridStatement

This statement adds or removes database entries.

Syntax:

[92] AlterGridStatement ::= ALTER GRID GridName
{ ADD | DROP } [DATABASE] DatabaseName

Explanation:

- *`ALTER GRID GridName ADD DatabaseName`* adds database DatabaseName to grid Gridname.
- *`ALTER GRID GridName DROP DatabaseName`* removes database DatabaseName from grid Gridname. The database itself remains unaltered.

```
ALTER GRID grid1 ADD DATABASE db4
ALTER GRID grid1 DROP db2
```

A.18. DropGridStatement

This statement deletes a database grid.

Syntax:
[93] DropGridStatement ::= DROP GRID GridName

Explanation:

- *`DROP GRID GridName`* deletes a grid GridName

```
DROP GRID grid1
```

Appendix B. The sysdatabase(s) Table

The sysdatabase table contains entries for all configuration parameters of a database.

The sysdatabases table in the admin database contains entries for all configuration parameters of all databases managed by the corresponding Transbase service.

Field	Explanation
database_name	The name of the database concerned. [sysdatabases only]
property	The database property or database configuration parameter.
value	The current value of this property for the specified database.
unit	The unit in which the current value is specified where this applies.
datatype	The data type of the property.
comment	Gives useful information on this property. Mostly the collection of possible values and the default value.