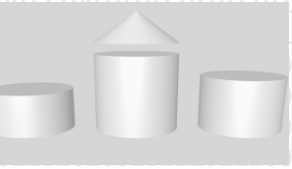

ODABA^{NG}

Replication Database

run Software-Werkstatt GmbH
Weigandufer 45
12059 Berlin

Tel: +49 (30) 609 853 44
e-mail: run@run-software.com
web: www.run-software.com

Berlin, October 2012

Content

1 Introduction 4
ODABA2 4
Platforms 4
Interfaces 4
User Interfaces 4

2 Replication Database 5
Master 5
Clients 5
Synchronize Database 5
Enable/disable 6
Limitations 6

Database Synchronization 7
Backup database 7
Transaction limit 7
Force reload 7

Transactions and locking 8
Throughput 8
Concurrency 8
Locking 9
Temporary Collections 9

Replication bridge 11
Local replication clients 11
File server 11
Object server 11

1 Introduction

ODABA2

ODABA2 is an object-oriented database system that allows storing objects and methods as well as causalities. As an object-oriented database, ODABA2 supports complex objects (user-defined data types), which are built on application relevant concepts.

ODABA2-applications are characterised by a high flexibility that is achieved by supporting in addition to object (concept) hierarchy, multifarious relations between objects (master and detail relations, relations between independent objects and others). This way conditions and behaviour of objects in the real world can be represented considerably better than in relational systems.

ODABA2-applications cannot only be drawn up as event-driven applications within the field of the graphical surface but also at the database level. This is one more way in which the application design is very close to the problem.

This makes ODABA2-applications a favourite possibility to solve highly complex jobs as come up in administrative and knowledge areas.

Platforms

ODABA2 supports windows platforms (Windows95/98/Me, Windows NT and Windows 2000) as well as UNIX platforms (Linux, Solaris).

You can build local applications or client server applications with a network of servers and clients.

Interfaces

ODABA2 supports several technical interfaces:

- C++, COM as application program interface (this allows e.g. using ODABA2 in VB scripts and applications)
- ODBC (for data exchange with relational databases)
- XML (as document interface as well as for data exchange)

User Interfaces

ODABA2 provides special COM-Controls that easily allow building applications in Visual Basic. On the other hand ODABA2 provides a special ODABA2 GUI builder based on the QT framework.

2 Replication Database

Replication database in ODABA is a feature, that not only allows creating a database replicate for security reasons, but also supports client in slow network environments as the internet.

Replication clients are the best solution, when running the server in a slow network (e.g. running a server in the internet). Replication clients may become inefficient when the client has intensive write access to the server, as for importing large amount of object instances.

Master

Within a simple replication environment, there exists one replication master, which has access to the replication master database. The replication master will synchronize data for all connected clients, i.e. it always keeps clients up-to-date.

Each transaction the replication master receives from its clients is sent to all connected clients. It is also stored in a transaction cache. Thus, clients can also be updated, when they have been offline for a while.

Clients

Replication clients are applications, which run somewhere in the network. Replication clients do have a local database replicate, which is accessed by the application.

Synchronize Database

When starting up a replication client the first time, the client will download a database backup from the replication master. This will also happen, when the replication client is outdated.

A replication client is outdated, when the version of the client database does nit correspond to the server version.

When starting a replication client the next time, the client is synchronized with the server by loading all missing transactions, i.e. all transactions registered after the last client session. While running the replication client, the local database copy is updated always, when starting a read transaction, i.e. when requesting data from the database.

Updates made by the client are sent to the replication master immediately after finishing a transaction (internal or user transaction). The server updates the master database and sends a dirty event to all other clients con-

nected.

Enable/disable

Once, a database has been enabled for replication, it can be disabled for being used again in a local environment. Since databases enabled for replication cannot be used in local environment in parallel, replication databases will automatically be disabled when being used in a local environment.

After using the database in local mode, it cab be re-enabled for replication.

Timestamp

Always when a database, which in not enabled for replication, will be enabled, the replication database gets a time stamp, which is used later on for comparing database versions.

Version

A database version is set in addition to the time stamp for identifying the database. This version is set to 0 when the database has been time stamped. You may explicitly increase the replication version by re-enabling an enabled replication database.

Limitations

In the current version, running replication server may conflict with other ODABA features.

Workspace

Replication servers do not work properly when the workspace feature has been enabled for the database.

Local clients

Each client for a replication database needs its own database copy, i.e. each application running on a replication database must have its own replicate, also when running on the same machine.

In one of the next versions this limitation will be removed and database replicates can be shared in a local network.

Database Synchronization

Backup database

For updating new or outdated clients, the replication database is loaded as backup from the replication server. The backup must be located on a (WEB) folder, which is accessible for all clients.

Usually, the backup will not represent the latest state of the replication database, since several clients may have updated the database after the backup has been made.

Hence, after downloading the backup, the client tries to synchronize the database state with the replication master. This can work properly, only when the database backup has the same version and the same timestamp as the current replication master database.

In general, it is suggested to run a service, which creates a database backup each day (or night) to reduce the server burden at run-time.

Transaction limit

When the number of transactions becomes very large, starting up an application may take some time. Hence, the number of transaction stored for synchronisation can be limited (default: 5000). The replication master will remove transactions from the cache, which are below the maximum number of transactions to be transferred.

When a replication client starts that cannot be updated by the transactions registered (i.e. the difference between the last transaction number the client received and the current transaction number exceeds the transaction limit), the client is considered as outdated and a backup of the database is sent to the clients.

Hence, the database backup, which can be downloaded from the replication server, must never be outdated or nearly outdated, because clients may fail to start up in this case.

Force reload

To force clients to reload the latest backup from the database (e.g. because of an error correction), the replication database must get at least a new version by explicitly re-enabling it.

Transactions and locking

When running replication clients, write transactions are serialized. This is necessary to guarantee database consistency.

Practically it means, that a client starting a write transaction (internal or user transaction) will block the replication master against other write transactions, i.e. other clients have to wait, until the first client has finished its transaction.

Throughput

Most transactions in ODABA applications are internal transactions, which take about 1 millisecond on the server, but taking into account, that communication requires about 50 milliseconds in addition, the limits of the replication master become obviously. Thus, under ideal conditions, the replication server can process about 20 transactions per second.

To avoid unnecessary communication overhead, sequences of modifications should be enclosed in transactions. User transactions will reduce the number of communications to 2 (start and stop transaction).

Also Critical is the use of long user transactions, since each transaction will block the server as long as it is active. Thus, user transactions should not exceed a few seconds and should never require user interactions during the transaction. In this case, users may block the whole system.

Concurrency

The pessimistic locking algorithm prevents the applications from producing database inconsistencies. Since each client will synchronize its database before starting the transaction it is sure, that the client database state corresponds to the master database, when running a transaction.

In some cases, however, the client receives new data from the server for the instance, which is just under processing, since it has been updated by another client. Since ODABA starts the internal transaction, just before going to store changes made on an instance, i.e. after all changes have been made on the internal instance based on the state the instance had when being read.

This is not only a replication database problem, since it happens also for file server or ODABA server applica-

tions. The system detects the difference for the instance version and refuses storing the changes.

Locking

To avoid concurrency conflicts, you may lock the instance explicitly before going to update it. There is an essential difference to file server or object server applications, which prevents all other applications from using an instance being updated. Locking for a replication server works only when all client applications will explicitly lock instances being used.

Write protection

ODABA provides an implicit pessimistic locking feature for file server or object server applications by accessing instances in write mode (instead of update mode). This works similar for the replication server, but it might be time consuming, since each internal lock may require between 30 and 100 milliseconds on the client, which corresponds to the communication time between client and server.

When using implicit write locking, each instance is blocked on the server against write requests from other clients. But there is still a difference to file or object server applications. While file or object server check any instance (update instances included) for internal write protection, the replication server only checks explicit write requests.

Tranferring write locks

In general, it would be possible to transfer write locks among the clients, but this is not yet implemented for the replication server. The question is still, whether such a feature is practical useful or not. Most applications do not use implicit write locks, since it strongly restricts the accessibility for the data. Thus, it seems more appropriate to take the risk of a failing transaction, which is usually restricted to a single instance.

Indexes

Indexes and collections are not affected by the locking problem, since those are updated within internal transaction, only. Applications do never change index states directly as they do for instance states.

Temporary Collections

In many cases, query results are stored in temporary collections. Updates or instances written to temporary collections need not to be transferred to the replication server. Since the transaction running does not know, which resources will be updated, starting a transaction will always submit a start transaction message to the replication master.

	Even though no data is transferred to the replication master in case of updating temporary resources, only, each create or add to the temporary collection will send a Begin- and CommitTransaction() to the replication server. Thus, each transaction takes about 100 ms in addition, while the real transaction time is perhaps about 1 ms.
DisableWrite	To avoid unnecessary communication with the replication master, you may temporarily disable the write feature for the database handle, which still allows writing to temporary resources but not to the “real” database.

Replication bridge

A replication client may act again as server. Thus you may build a hierarchy of replication servers with a replication master on top. This is a theoretical possibility and has not yet been tested.

It might help, when clients are distributes all over the world to improve the performance.

So far, it has not yet been discussed, how replication bridges could work in a network. This we have left for the future.

Local replication clients

A more interesting use case is running a local replication client as file or object server e.g. within the local network of an organization. This makes sense especially, when having a larger number of clients working with the database (more that 10 or 20).

In this case the traffic between the replication clients and the master is reduced extremely, since instead of 20 only one replication client needs to be synchronized.

Another positive aspect in such an environment is, that within the local network implicit write locks are fully supported, i.e. the risk of update conflicts is reduced.

File server

When running the replication client as file server, each client reports its transactions to the replication master. The database is synchronized by the first client, which detects that the database is not up-to-date. The other clients may receive a dirty event, but in most cases there will be nothing new to be updated.

Object server

When running the replication client as object server, the object server is the only client communicating with the replication master. In this case, communication is reduced to the communication between object server and replication master. Internally, all clients communicate only with the object server.