Hochschule Mittweida (FH)
University of Applied Sciences

DIPLOMARBEIT

Markus Rüf

Produktionsdatenbanksynchronisierung mit benutzergeführter Änderungsübernahme

2011

Hochschule Mittweida (FH)
University of Applied Sciences

Markus Rüf

Produktionsdatenbanksynchronisierung mit benutzergeführter Änderungsübernahme

eingereicht als

Diplomarbeit

an der

HOCHSCHULE MITTWEIDA

UNIVERSITY OF APPLIED SCIENCES

Fachbereich Informationstechnik & Elektrotechnik

Lustenau, Mai 2011

Erstprüfer:	Prof. Dr. rer. biol. hum. Rudolf Stübner
Zweitprüfer:	Prof. Dr. Ing. Uwe Schneider

Vorgelegte Arbeit wurde verteidigt am: 04. Juli 2011

Bibliografische Angaben

Rüf, Markus:
Produktionsdatenbanksynchronisierung mit benutzergeführter Änderungsübernahme

115 Seiten, 32 Abbildungen, 14 Tabellen, 6 Anlagen mit SQL Source Skripts,
Hochschule Mittweida (FH), Fachbereich Informationstechnik & Elektrotechnik
Diplomarbeit, 2011

Kurzreferat:

Viele Anwendungen greifen heute für ihre Arbeit auf Datenbanken zu. Oft ist es erforderlich, diese Datenbanken miteinander zu synchronisieren. Die Arbeit beschäftigt sich mit den verschiedenen Replikationsverfahren in der Theorie, um dann eine konkrete Lösung zu entwerfen zur Synchronisierung von mehreren Produktions-Datenbanken in einem Unternehmen. Die erarbeitete Lösung wird im Detail beschrieben und ihre Funktionalität erklärt.

Abstract:

Today many applications use databases in the background. Often it is necessary to synchronize those databases with each other. The work explains the various replication methods in the theory and in the second part, a solution for synchronizing multiple databases in a production company is designed. The developed solution will be described in detail and its functionality will be explained.

Inhaltsverzeichnis

Inhaltsverzeichnis i
Abbildungsverzeichnis v
Tabellenverzeichnis vii
Abkürzungsverzeichnis ix
Sperrvermerk i
Vorwort ii
1 Einleitung 1
1.1 Motivation 1
1.2 Problemstellung 2
1.3 Zielsetzung 5
1.4 Gliederung 6
2 Grundlagen 7
2.1 Datenbanken 7
2.2 Datenbankmanagementsystem und Datenbanksystem 8
2.3 Geschichte der Datenbanken: von der Stapelverarbeitung über die Dateiverwaltung zu heutigen Datenbanksystemen 9
2.4 Anforderungen und die Vorteile eines modernen Datenbanksystems 10
2.5 Der Datenbankentwurf 12
2.6 Datenbankmodelle 14
2.7 Entity-Relationship-Modell 16
2.8 Relationales Datenmodell 18
2.8.1 Begriffe der relationalen Datenbank, vom Kleinen zum Grossen 20
2.9 Vom ERM zum relationalen Datenmodell 23
2.9.1 Transformation und Umsetzung des ERM in ein Relationen Modell 23
2.9.2 Normalisierung 25
2.10 Datenbank Abfragesprache SQL 27
2.11 Verteilte Datenbanken 28
2.12 Zusammenfassung 28
3 Replikation in verteilten Datenbanken 29
3.1 Abgrenzung: Verteilte Systeme und Parallelität 29
3.2 Anwendungsszenarien von Datenbankverbunden 29
3.3 Replikation 30
3.3.1 Definition Replikation 30
3.3.2 Vorteile der Replikation 31
3.3.3 Anwendungsszenarien für die Datenbankreplikation 32
3.4 Ablauf der Replikation 33

3.5 Zielkonflikte ... 34
3.6 Replikationsstrategien/ Replikationseinteilung ... 35
3.7 Replikationskonflikt ... 38
3.8 Replikationsverfahren ... 38
3.8.1 Klassische, konsistenzerhaltende Verfahren ... 38
3.8.2 Verfügbarkeitserhaltende asynchrone Verfahren ... 40
3.9 Zusammenfassung ... 42
4 Anforderungen ... 43
4.1 Aktueller Stand der vorhandenen Software ... 43
4.2 Problemstellung und Kundenforderung ... 45
4.3 Weitere Anforderungen ... 45
4.4 Derzeit bekannte Randbedingungen ... 46
4.5 Beschreibung der vorhandenen Systeme ... 46
5 Konzept ... 47
5.1 Replikationsverfahren im MS SQL Server ... 47
5.2 Begriffe für die Replikation mit dem MS SQL Server ... 48
5.2.1 Verleger ... 49
5.2.2 Verteiler ... 49
5.2.3 Abonnenten ... 49
5.2.4 Transaktionsreplikation ... 50
5.2.5 Mergereplikation ... 51
5.2.6 Snapshotreplikation ... 52
5.3 Kriterien zur Entwurfsentscheidung ... 53
5.3.1 Synchronisierungsrichtungen ... 53
5.3.2 Synchronisierungsmethoden/-funktionen ... 53
5.3.3 Aufteilung der Synchronisierung: ... 53
5.4 Konzeptentwurf RZNW-MultiServer ... 54
5.4.1 Kommunikation HS → STAO ... 55
5.4.2 Kommunikation STAO → HS ... 55
5.5 Funktionsweise ... 57
5.5.1 Rezept-Vorlagen ... 57
5.5.2 Allgemeine Maschinenparameter ... 59
5.5.3 Maschinenverwaltung ... 59
5.5.4 Rezepte ... 59
5.5.5 Logbücher ... 60
6 Realisierung ... 61
6.1 Notwendige Datenbank-Erweiterungen ... 61
6.2 Konfigurationshinweise für die Replikation ... 67
6.3 Erzeugen von RZNW-Schnappschüssen am Stao ... 69
6.3.1 Export-DB ... 69
6.3.2 Ablauf der Datenübertragung ... 69
6.3.3 Schnappschuss der RZNW-DB erzeugen ... 69
6.3.4 Snapshotgenerierung über Agenten per SQL starten ... 70

6.3.5 Distribution Agenten per SQL starten ... 71
6.4 Handshake Stao → HS ... 71
6.4.1 Sync_Log ... 71
6.4.2 Notification ... 74
6.5 Übernahme der Standort-Daten am Hauptserver ... 76
6.5.1 Zugriff auf die Haupt-DB ... 76
6.5.2 Stored Procedures ... 76
6.5.3 Effektive Übernahme ... 77
6.5.4 Geänderte Vorlagen ... 77
6.6 Erzeugen von RZNW-Schnappschüssen am Hauptserver ... 78
6.6.1 Export-DB ... 78
6.7 Übernahme der Hauptserver-Daten am Standort ... 78
6.7.1 Vorlagen ... 79
6.7.2 Rezepte ... 79
6.7.3 Maschinenverwaltung ... 79
6.7.4 Stored Procedures ... 80
6.7.5 Effektive Übernahme der Daten vom HS in die STAO Datenbank ... 80
7 Zusammenfassung und Ausblick ... 81
Anlagen ... 83
Anlage 1 ... 84
Anlage 2 ... 85
Anlage 3 ... 87
Anlage 4 ... 99
Anlage 5 ... 102
Anlage 5 ... 110
Literaturverzeichnis ... 112
Erklärung ... 115

Abbildungsverzeichnis

Abb. 1: gebaute PKW- und LKW Nockenwellen der TKT-Presta 2
Abb. 2: Anwendersoftware RZNW mit geöffnetem Editor für die Erstellung der Produktionsrezepte 3
Abb. 3: Vorlagen Editor zum Pflegen des Masterrezeptes 4
Abb. 4: Microsoft SQL Server Management Studio mit rznw-stao1 RZNW Datenbank 4
Abb. 5 Datenbanken und Benutzergruppen 7
Abb. 6: Vereinfachtes Modell einer Datenbank-Umgebung 8
Abb. 7 Einsatzgebiete der Datenverwaltungs- und Datenbanksysteme 10
Abb. 8 Aufgaben/Vorteile eines Datenbank-Management-System 11
Abb. 9: Aufgaben/Vorteile eines Datenbank-Management-System 12
Abb. 10: vereinfachtes Vorgehensmodell zur Datenbankentwicklung 13
Abb. 11: hierarchische und netzwerkartige Datenbanken 14
Abb. 12: hierarchische Datenmodell 15
Abb. 13: relationales Datenmodell, ERD von Teilen des RZNW-Datenbankschemas 16
Abb. 14: Eine Beispiel-ER-Modellierung 17
Abb. 15: relationale Beziehungen 18
Abb. 16: Fachbegriffe für Tabellenbestandteile 20
Abb. 17: Das Replikationsablauf 31
Abb. 18: Zielkonflikt der Replikationskontrolle 34
Abb. 19: Klassifikation der Replikationsverfahren 36
Abb. 20: Klassifikation der Replikationsstrategien 37
Abb. 21: RZNW-Struktur Landschaft an einem Standort 43
Abb. 22: RZNW-Vorlageneditor für die Konfiuration des Master (Maximal)-Rezepts 44
Abb. 23: RZNW-Rezeptur Strukturierungselemente 44
Abb. 24: Komponenten und Prozesse die bei der Replikation beteiligt sind 48
Abb. 25: Ablauf der Transaktionsreplikation 50
Abb. 26: Ablauf der Merge-Replikation 51
Abb. 27: Ablauf der Snapshot-Replikation 52
Abb. 28: Konzept RZNW Multiserver 54
Abb. 29: funktioneller Ablauf der Vorlagensynchronisierung 58
Abb. 30: alle am HS erstellten LocImport und MainExport Datenbanken 67
Abb. 31: STAO alle Publikationen zum HS und die Subscriptions auf HS Publikationen 68
Abb. 32: Job Profile eines Agenten: Feintuning at its best 68

Tabellenverzeichnis

Tabelle 1: 1NF 25
Tabelle 2: nicht in 1NF.....25
Tabelle 3: Studenten-Projektarbeiten nicht in 2NF.....26
Tabelle 4: ausgewählte SQL Implementierungsversionen 27
Tabelle 5: Merge Publikationen 56
Tabelle 6: neue Tabellen 61
Tabelle 7: Schnappschussgenerierung StoredProcedures am STAO.....70
Tabelle 8: Snapshot Agenten per StoredProcedures am STAO ausführen.....71
Tabelle 9: Attributbeduetung der Sync_Log Tabelle 72
Tabelle 10: Attributbeduetung der LocalModifiedElements Tabelle der Notification Publikation 75
Tabelle 11: Skripte für die Datenübernahme-von der Import DB in die HS DB 76
Tabelle 13: Schnappschusserstellungsscript am HS 78
Tabelle 14: Schnappschussanwendungscript am STAO 80

Abkürzungsverzeichnis

Abb.	Abbildung
Aufl.	Auflage
bzw.	beziehungsweise
ca.	circa
d.h.	das heisst
DBMS	Datenbankmanagementsystem
DE	Deutschland oder Deutsch
DV	Datenverarbeitung
EDV	elektronische Datenverarbeitung
erw.	erweitert(e)
etc.	et cetera
EUR	Euro
ff	folgend(e)
FL	Liechtenstein
ggfs.	gegebenenfalls
Hrsg.	Herausgeber
HS	Hauptserver oder Hauptserverdatenbank
IT	Informationstechnologie
Mio	Million(en)
MS	Microsoft
ODBC	Open DataBase Connectivity
RZNW	Rezeptur Verwaltung Nockenwelle
SPS	speicherprogrammierbare Steuerung
SQL	Structured Query Language
SRV	Server
STAO	Standord oder Standort Server oder StandortDatenbank
SW	Software
Tab.	Tabelle
TKT	ThyssenKrupp Automotive
TKT-Presta	ThyssenKrupp Presta Camshafts
u.a.m.	und andere(s) mehr
überarb.	überarbeitet(e)
usw	und so weiter

vgl.	vergleiche
Vgl.	Vergleich
WAN	wide area network
wesentl.	wesentlich
z. B.	zum Beispiel

Sperrvermerk

Auf Wunsch der Firma ThyssenKrupp Presta Camshafts ist die vorliegende Diplomarbeit für die Dauer von zwei Jahren für die öffentliche Nutzung zu sperren.
Einsichtnahme, Veröffentlichung und Vervielfältigung sind ohne ausdrückliche Genehmigung der Firma ThyssenKrupp Presta Camshafts und des Verfassers bis zum 30. Mai 2013 nicht gestattet.

Lustenau, 10. Mai 2011

Hinweis: Diplomarbeitsthema sowie die Kurzreferat/Abstract dürfen veröffentlicht werden.

Vorwort

Die vorliegende Diplomarbeit wurde in der Zeit von Dezember 2010 bis Mai 2011 bei der ThyssenKrupp Presta Camshafts in Eschen, Liechtenstein ausgearbeitet.

Mein besonderer Dank gilt Herrn Prof. Dr. rer. biol. hum. Rudolf Stübner für die Betreuung und Beurteilung der vorliegenden Diplomarbeit an der Hochschule Mittweida. Bedanken möchte ich mich auch bei all jenen, die mich in irgendeiner Art und Weise unterstützt haben.

Anmerkungen

Aus Gründen der besseren Lesbarkeit kommen in dieser Arbeit für beide Geschlechter die grammatikalisch maskulinen Bezeichnungen zur Anwendung. Sind bei den Abbildungen oder Tabellen keine Quellen angegeben, wurden diese vom Verfasser erstellt. In der Schweiz wird statt ’ß’ immer ’ss’ geschrieben.

1 Einleitung

1.1 Motivation

Die Informatik bzw. Datenverarbeitung ist bis zum heutigen Tag von raschen Technologiewechseln geprägt.
Diese Dynamik entsteht nicht nur durch stetig geänderte organisatorische Abläufe und Anforderungen, sondern vielmehr aus den technologischen Entwicklungen. Wegen der Zunahme der technologischen Änderungsgeschwindigkeit und immer häufiger auftretende Systemwandel in kürzerer Zeit, müssen gerade auch Software-Anwendungen und die Hardware auf denen sie ausgeführt werden immer rascher angepasst werden. Daraus ergab sich in den letzten Jahren, dass sich der verteilte Systemansatz immer mehr durchgesetzt hat, im Gegensatz zur Zentralrechner Architektur in den achtziger Jahren des letzten Jahrtausends. Diese neue Rechnerlandschaft wird auch von wirtschaftlichen Überlegungen getrieben, weil kleinere Rechner durch deren höhere Stückzahlen oft kostengünstiger in ihrer Anschaffung, ihrer Wartung und ihrem Unterhalt als zentrale Mainframe Rechner sind und viel besser auf ihren jeweiligen Einsatzzweck hin skaliert werden können. Diese Entwicklung wurde auch durch immer leistungsfähigere und günstigere Kommunikationsnetze beschleunigt. Aus diesen Gründen werden heutzutage auch für sehr leistungsintensive Anforderungen verteilte Systeme gegenüber den zentralen Systemen bevorzugt.
Die Systemlandschaft ist heute aber auch deshalb verteilt, weil das Internet durch die Zunahme mobiler Endgeräte wie mobile Rechner, Laptops, Smartphones usw. in fast alle Lebensbereiche vor- und eingedrungen ist.
Kamen früher Datenbanken vor allem bei grossen Daten-Katalogen und bei der Verarbeitung von wirtschaftlichen Daten (z.B. Kontoführungen, Kennzahlensysteme) zum Einsatz, so ist durch deren Weiterentwicklung zu objektbasierten Datenbanken ihr Einsatzgebiet rasant angestiegen. Sie spielen heute die zentrale Rolle in der Datenhaltung, aber nicht als einzelne, grosse zentrale Datenbank, sondern als verteilte, oft autonom operierende Datenbanken, und erst die Anwendersoftware bildet aus den einzelnen, verteilten Daten ein wohlgeformtes Ganzes für den Benutzer.
Datenbanken spielen auch in der industriellen Produktion von Gütern schon länger eine wesentliche Rolle. Und zwar nicht nur um wirtschaftliche Kennzahlen und Kenngrössen aus historischen Daten zu gewinnen, sondern um unmittelbar die Produktionsmaschinen mit den richtigen Parametern zu versorgen. Der Herstellungsprozess, der auf diesen Maschinen abläuft, wird meist mit so genannten Rezepturen beschrieben.

Eine Rezeptur beinhaltet typischerweise die Bearbeitungsschrittfolgen, die dazu verwendeten Stellparameter für z.B. Aufpresskräfte, Werkzeugparameter, Parametergrenzen bzgl. Qualität und der eingesetzten Maschinen und Werkzeugen, sowie ökonomische Daten wie Auftragsstückzahl usw.

1.2 Problemstellung

Die ProgTec GmbH in Altstätten, Schweiz ist der Hersteller eines solchen rechnergestützten Rezepturverwaltungsprogrammes namens 'Rezeptur Datenverwaltung für Nockenwellen', kurz RZNW für die Firma ThyssenKrupp Presta Camshafts, kurz TKT-Presta. Die TKT-Presta mit ehemaligem Hauptsitz in Eschen Liechtenstein, gehört zur Automotive Sparte des weltweit tätigen ThyssenKrupp-Konzerns und produziert sehr erfolgreich „gebaute Nockenwellen" für verschiedene Automobilhersteller. Diese Nockenwellen werden im Verbrennungsmotor verwendet um die Ventile beim Verbrennungsvorgang präzise zu öffnen und zu schliessen. Vereinfachend erklärt bedeutet „gebaute Nockenwellen", dass auf ein Zentralrohr die Nocken und andere Bauteile in der richtigen Art und Weise aufgepresst werden. Das ist erheblich schneller und auch viel günstiger in der Herstellung, als gegossene Nockenwellen.

Abb. 1: gebaute PKW- und LKW Nockenwellen der TKT-Presta[1]

Die TKT-Presta hat weltweite Niederlassungen in Eschen/FL, Dalian/China, Danville/USA, Chemnitz/DE und Ilsenburg/DE. Es sind ca. 1020 (Stand 2008) Mitarbeiter beschäftigt, bei einem Jahresumsatz von ca. > 200Mio. EUR (Stand 2008).[2]

[1] Vgl. /TKT-Presta Info/

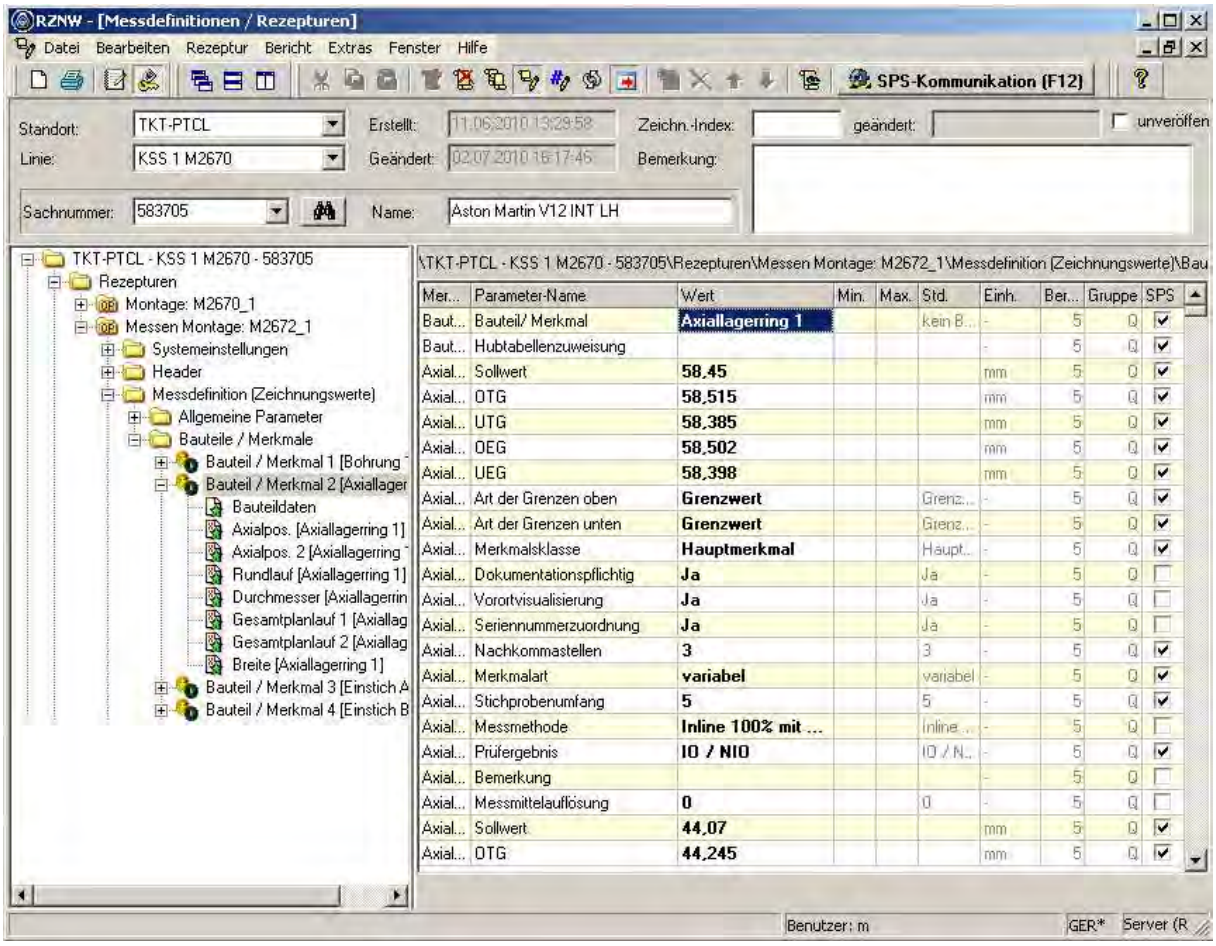


Abb. 2: Anwendersoftware RZNW mit geöffnetem Editor für die Erstellung der Produktionsrezepte

Mit der RZNW Software können Produktrezepturen erstellt und gepflegt werden. Für die Produktion kann dann von der RZNW aus, das Rezept über unterschiedliche Kommunikationsprotokolle zu den verschiedenen SPS[3] Typen geladen werden. Damit die Rezepturen nicht willkürlich und ihre Struktur auch für unterschiedliche Maschinen immer nach vorgegebenen Mustern erstellt werden, wird über den Vorlagen-Editor in der RZNW ein Master Rezept erstellt. In den Vorlagen werden alle nur möglichen Prozessausprägungen, seine Bauteile und sein Herstellungsvorgang über Parameter definiert. In einem konkreten Produkt-Rezept[4] können dann aus diesen Vorlagen die benötigten Elemente ausgewählt werden, dass sie bestmöglich diesen einen Produktionsprozess beschreiben und abbilden. Eine Produktions-Anlage umfasst eine oder mehrere Maschinen, auf denen wiederum eine oder mehrere Operationen ausgeführt werden. Somit ist ein Rezept in die Beschreibung von den einzelnen Operationen der Maschine unterteilt.

---

[2] /TKT-Presta Info/

[3] SPS, Abkürzung für speicherprogrammierbare Steuerung oder englisch PLC, Programmable Logic Controller

[4] 'Rezept' und 'Rezeptur' werden synonym verwendet

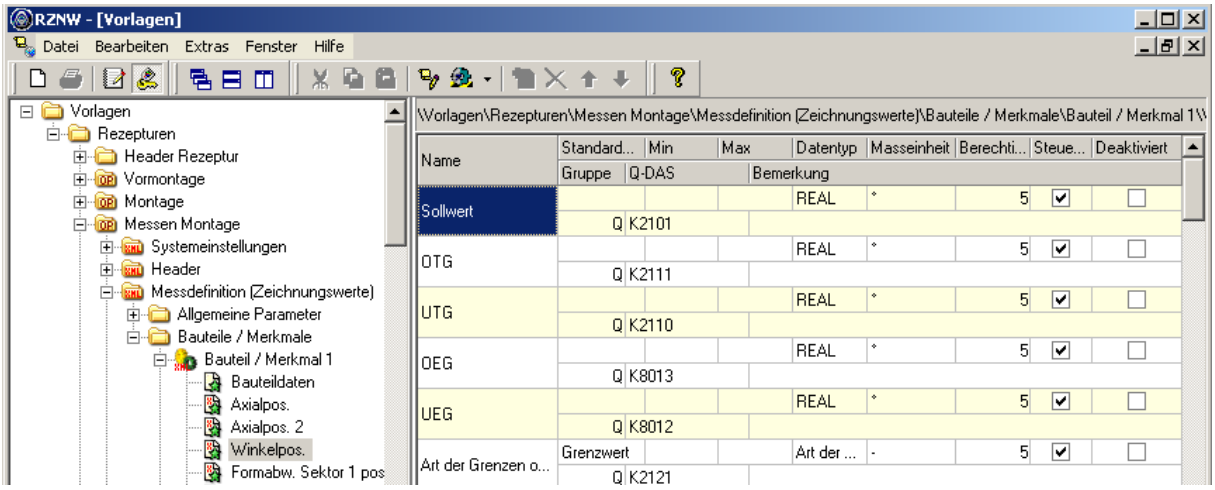


Abb. 3: Vorlagen Editor zum Pflegen des Masterrezeptes

Für die Speicherung der Daten wie Parameter und der Arbeitsschrittabfolgen usw. verwendet die RZNW eine relationale Datenbank in einem Datenbankmanagementsystem (DBMS) von Microsoft SQL Server (es kommen die Server Versionen 2000, 2005 und 2008 R2 Standard Edition zum Einsatz).

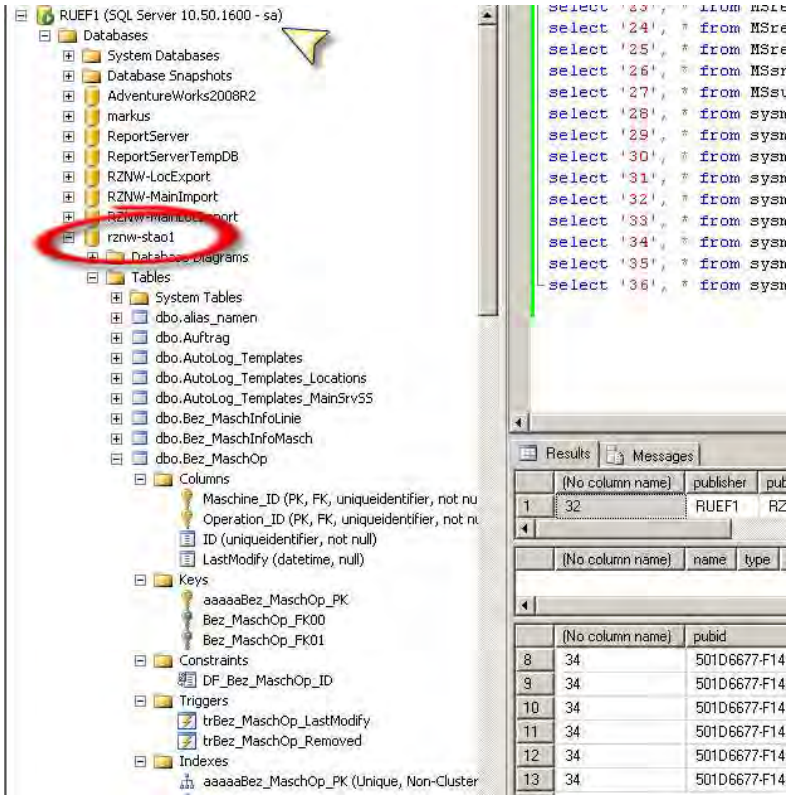


Abb. 4: Microsoft[5] SQL Server[6] Management Studio mit rznw-stao1 RZNW Datenbank[7][8]

[5] Microsoft, kurz MS

[6] SQL Server, kurz SQL Srv

[7] Datenbank: in dieser Arbeit werden sofern nicht anders erwähnt, immer relationale Datenbanken gemeint

[8] Datenbank, kurz DB

In der globalisierten Welt werden gleiche Produkte bzw. Produkttypen nicht mehr nur an einem Standort produziert, sondern an Vielen. Um die Qualität der Rezepte, die Wartung, die Dokumentation und nicht zuletzt die Rezeptureingabe effizient zu gestalten, besteht nun eine neue Anforderung diese RZNW Datenbanken an den Produktionsstandorten zu vernetzen Damit sollen die Rezepte eines Standortes auch auf anderen Standorten verfügbar sein und zwar so, dass fortlaufende Änderungen an den Rezepturen von einem Standort[9], kurz STAO auch zeitnah an anderen STAOs zur Verfügung stehen, d.h. ihre Struktur und die Werte der Parameter synchron sind. Da die Rezepte auf den Vorlagen basieren müssen die Vorlagen ebenfalls überall synchron sein. Vorlagenänderungen sollen an allen Standorten von geschultem und berechtigtem Personal durchgeführt werden. Diese Änderungen sollen an einer zentralen Datenbank zusammengeführt werden. Dort werden sie auf Korrektheit geprüft und dann auf alle anderen Standorte weiterverteilt. Dazu müssen Teile der Standortdatenbanken miteinander synchronisiert werden. Der primäre Begriff für die Synchronisierung von verteilten Datenbanken ist die Replikation. Es soll somit eine möglichst einfach zu bedienende, fehlertolerante und wartungsarme Softwarelösung basierend auf den derzeit eingesetzten Tools und Datenbanken entwickelt werden, um die Server Datenbanken von derzeit fünf Standorten der TKT-Presta miteinander zu synchronisieren.

1.3 Zielsetzung

Das Ziel dieser Arbeit ist es einen Überblick über die 'State of the Art' Replikationsverfahren von relationalen Datenbanken zugeben und sie verständlich zu erläutern. Daneben soll unter Einbeziehung der realen Systemlandschaft und Architektur bei der TKT-Presta eine konkrete Replikations Lösung ausgearbeitet werden. Sie soll beinhalten, wie die verschiedenen Werksstandort-Datenbanken untereinander synchronisiert werden, so dass Änderungen an den STAOs über einen Hauptserver[10][11], auf die anderen Standorte repliziert werden können. Die Replikation ist notwendig damit zukünftig alle RZNW-Benutzer möglichst zeitnah die gleichen Informationsstände vor Ort zur Verfügung haben. Der konkrete Nutzen dieser Entwicklung für die TKT-Presta soll sein, eine vereinheitlichte Datenbasis zu schaffen um den derzeit vermehrt vorherrschenden Wildwuchs an den Vorlagenstrukturen an den Standorten abzuschaffen, und somit die Qualität insgesamt deutlich anzuheben.

[9] Standort: mit Standort wird immer ein zuvor erwähnter Produktionsstandort der TKT-Presta gemeint

[10] Hauptserver: als Hauptserver ist eine normale zentrale RZNW Datenbank gemeint, an der zukünftig alle Änderungen zusammenfliessen

[11] Hauptserver, kurz HS

„Es soll nicht nur geforscht und evaluiert werden, sondern am Ende muss eine den Anforderungen entsprechende und wirtschaftlich umsetzbare Lösung vorliegen.“ (Joel Bauer, Projektleiter und Auftraggeber bei der TKT-Presta).

1.4 Gliederung

Die Arbeit ist deshalb in **zwei** Teile geteilt.
Im **ersten** Teil soll die **theoretische Fundierung** zu den einzelnen Datenbankkonzepten und insbesondere den Replikationsverfahren erläutert werden.
Im **zweiten** Teil soll ein **Lösungsansatz** entwickelt werden, dass auf einem Replikationsverfahren basiert, welches am Besten die Anforderungen des Kunden erfüllt.
Die zentralen Anforderungen sind:

- Performanz und Dauer der Synchronisierung,
- die Konsistenz der Daten auf allen Standorten, und mögliche auftretende Daten-Konflikte,
- technische Probleme und Fehler und
- eine praktische konkrete Umsetzung und Implementierung in der vorhandenen Systemlandschaft.

Kapitel 1 gibt eine kurze Einführung in das Thema der Diplomarbeit, indem die Motivation, die Problemstellung und die Ziele dargelegt werden.
Kapitel 2 führt in die Grundlagen der Datenbanken, ihrem Aufbau und ihrem Entwurf ein
Kapitel 3 stellt die verschiedenen Replikationsverfahren kurz vor
Kapitel 4 definiert die Kundenforderungen für die konkrete DB-Synchronisationslösung
Kapitel 5 stellt das ausgearbeitete und evaluierte Konzept vor
Kapitel 6 beschreibt die Realisierung der Lösung und gibt Einblick in einzelne Entwicklungsdetails und beschreibt die konkrete Lösung.
In **Kapitel 7** wird eine Zusammenfassung über die Arbeit gegeben, die Lösung kurz bewertet und mögliche weitere Entwicklungsschritte für die Lösung aufgezeigt.

2 Grundlagen

Dieses Kapitel beschreibt die theoretischen Grundlagen und Begrifflichkeiten die zum weiteren Verständnis der entwickelten Synchronisierungslösung dienen. Dazu werden zuerst einige allgemeine Datenbankbegriffe erklärt wie reale Objekte in einem Datenbankschema abgebildet werden, und was die Kennzeichen eines verteilten Systems sind.

2.1 Datenbanken

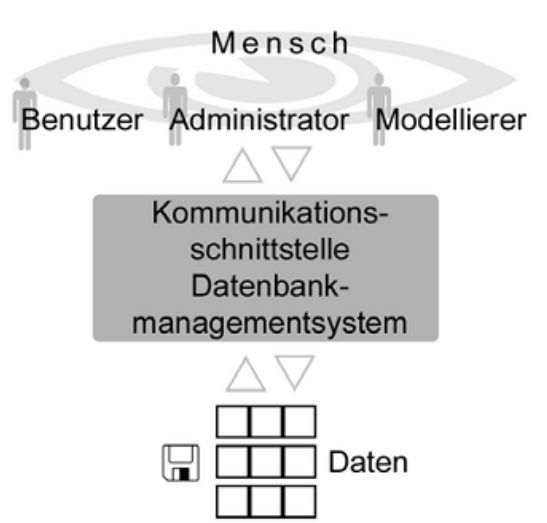


Abb. 5 Datenbanken und Benutzergruppen[12]

In der elektronischen Datenverarbeitung[13] werden Daten von einem Eingabegerät entgegen genommen oder durch ein Anwenderprogramm generiert, gespeichert, weiterverarbeitet, umgeformt, angepasst und hoffentlich als neue Information in der gewünschten Form aggregiert und konglomeriert. Für den Systembetrieb sind ebenfalls Daten notwendig, wie Benutzerrechte, Benutzerverwaltung, Systemdateien usw. Um diese Daten effizient zu verwalten und zu speichern werden Datenbanken verwendet.

Definition einer Datenbank

'Eine Datenbank ist eine Sammlung von Daten, die untereinander in einer logischen Beziehung stehen und von einem eigenen Datenbankverwaltungssystem (Database Management System, DBMS) verwaltet werden' [14]

[12] [Skulschus 07], p. 22

[13] In weiterer Folge mit EDV abgekürzt

[14] [Schicker, 96], p. 46

In Datenbanken werden Daten gespeichert. Diese sind ohne Zusammenhang noch relativ wertlos, erst durch ihre logische Verknüpfung mit anderen Daten und deren Einordnung oder Hierarchie mit anderen Daten und deren Bewertung gelangen wir zu den für den Anwender zweckdienlichen und nützlichen Informationen über Sachverhalte oder auch Vorgänge. Somit sind **Informationen interpretierte Daten** oder umgekehrt sind **Daten Teile von Informationen** die rechnergestützt verarbeitet werden können.

2.2 Datenbankmanagementsystem[15] und Datenbanksystem[16]

Ein Datenbankmanagementsystem ist ein Software-System zur Verwaltung von Datenbanken. Mit ihm können die Datenstrukturen und die Datenschemas innerhalb der Datenbank definiert und geändert werden. Es bildet nach aussen die Schnittstelle für die verschiedensten Anwendungen auf die Datenbank, um Daten zu speichern und aus ihr Daten wieder abrufen zu können.

Datenbanksystem = Datenbank (z.B. Kundenkonto) + Datenbankmanagementsystem (z.B. Microsoft SQL Server 2008 R2)

Ein Datenbanksystem ist der Oberbegriff für die Komponenten Datenbank und dem Datenbankmanagementsystem. Folgende Abbildung soll das verdeutlichen:

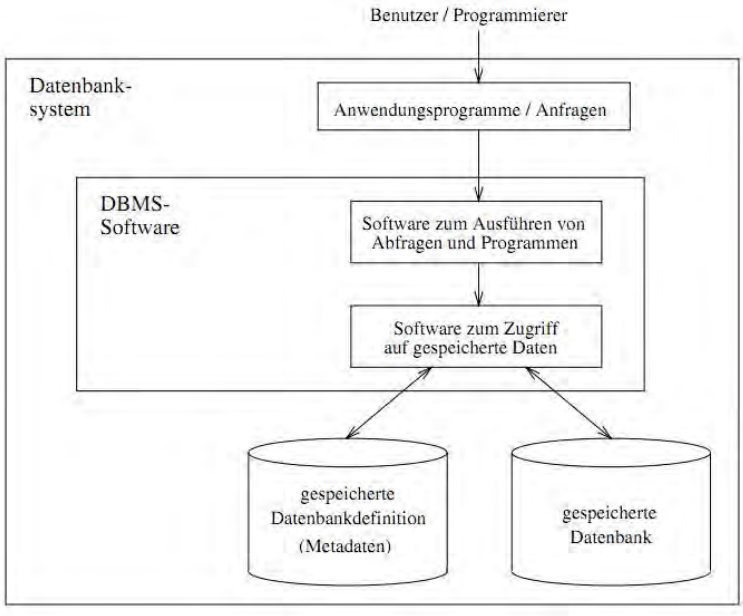


Abb. 6: Vereinfachtes Modell einer Datenbank-Umgebung[17]

[15] In weiterer Folge mit DBMS abgekürzt

[16] In weiterer Folge mit DBS abgekürzt

[17] [Klein 05], p. 4

Zu einem DBMS gehören neben den physischen Datenbankdateien und Logdateien vor allem die Metadatenverwaltung z.B. die Informationen über die Schemas, weiters die Transaktionsverwaltung, Recoverymechanismen für die Datensicherheit und auch ein Zugriffsrechtesystem.

2.3 Geschichte der Datenbanken: von der Stapelverarbeitung über die Dateiverwaltung zu heutigen Datenbanksystemen

Mit der Hilfe heutiger Datenbankkonzepte können nicht nur Daten abgespeichert werden, sondern auch Wissen. Bis hierhin war es aber ein langer Weg, und die Entwicklung von verschiedenen Datenbankkonzepten soll hier kurz umrissen werden.

Zu Beginn der elektronischen Datenverarbeitung konnten Daten in den Systemen nicht persistent vorgehalten werden, sondern waren sehr flüchtig und auf das jeweilige Programm zugeschnitten und die Entwicklung von diesen Individuallösungen deshalb auch sehr teuer. Der Zugriff auf die Daten erfolgte wie die Ausführung der Programme im Stapelbetrieb. Nach und Nach speicherte man die Daten in einer Art Dateisystem ab, aber der Zugriff erfolgte noch immer sequenziell. Die ersten Magnetplatten ermöglichten dann in folge den Zugriff über eine Adresse unabhängig der Speicherabfolge. Jedoch brachte dies immer noch keine Vereinfachung, sondern vielmehr Redundanzen und unzählbare Datenstrukturen, da jedes Programm seine eigenen Datentypen implementierte. Der Nachteil auch moderner Dateiverwaltungssysteme ist nach wie vor ihre Redundanz, geringe Effizienz und schwieriger Zugriffsschutz (z.B. in Baumstrukturen einen Unterordner eines geschützten Elternordners freigeben), Dateninkonsistenzen wegen fehlender Integritätsbedingungen (z.B. kann jeder seine Dateien benennen, wie er möchte). Ebenso wird der Datenaustausch durch die verschiedensten Darstellungsformen und Dateiformate erschwert (die bunte Welt der Dateiendungen). Standards lassen trotz Auszeichnungssprachen wie XML[18] noch auf sich warten.
Die ersten Datenbanksysteme entstanden in den 70er Jahren des vorigen Jahrhunderts und unterschieden schon in physischer und logischer Datenverwaltung. Diese so genannten **prärelationalen Datenbankensysteme** waren immer noch stark auf die jeweiligen Anwendungen zugeschnitten. Erst in den 80er Jahren wurden **relationale Datenbanksysteme** entwickelt die eine klare **Trennung** zwischen **physischer** Datenspeicherung und dem **logischen** Datenmodell und dem Zugriff auf die Daten vornahm. Sie ermöglichten erstmals die mengenorientierte anstatt der reinen Datensatzverarbeitung. Der kontrollierte und reglementierte

[18] Extensible Markup Language, englisch für 'erweiterbare Auszeichnungssprache'

Mehrbenutzerzugriff wurde durch diese frühen DBMS schon ermöglicht. Dies führte aber auch gleich zu einem grundlegenden Problem der Datenhistorie:
In einem Spalte/Attribut kann nur ein einziges Datum (Singular für Daten) stehen und es wird bei Mehrfachänderung einfach überschrieben (last update wins).
Mit dem Einzug von objektorientierten Programmiersprachen wurden auch objektorientierte Datenbanken entwickelt, die so genannten **postrelationalen** Datenbanken. Zu denen gehören auch verteilte und mehrdimensionale Datenbanken, die vor allem auch durch die immer leistungsfähigeren Netzwerke entstanden sind. Doch erst die heutigen Datenbanksysteme lassen Informationssysteme entstehen, in denen auch Wissen abgespeichert werden kann (neuronale Netze, semantische Cluster).

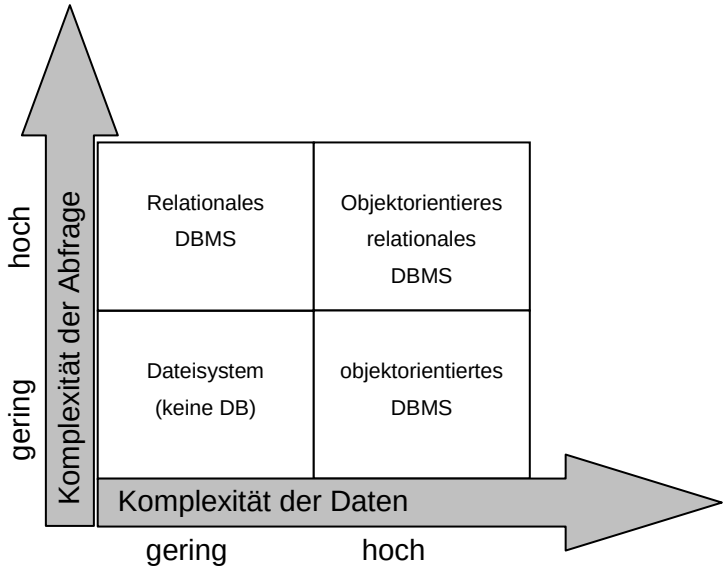


Abb. 7 Einsatzgebiete der Datenverwaltungs- und Datenbanksysteme

2.4 Anforderungen und die Vorteile eines modernen Datenbanksystems[19]

- hohe Datenunabhängigkeit (Zugriff)
  Die Anwendungen kommunizieren unabhängig von der Datenspeicherung und der Datenrepräsentation mit der DBMS
- Verwaltung der Daten mittels des zugrunde liegenden Datenmodels indem Bedingungen und Abhängigkeiten der Relationen bzw. Attribute untereinander definiert werden
- hohe Verfügbarkeit durch verteilte Systeme, Datenbanksicherungskonzepte und somit geringe Ausfallszeiten
- gute Skalierbarkeit und schnelle Anpassungsfähigkeit

[19] Vgl. [Skulschus 07], p. 20

- hohe Abfrage- und Speicher Leistung durch ausgewogene Wahl von Indexen
- hohe Fehlertoleranz durch Definierung von Integritätsbedingungen, die korrekte und vollständige Daten gewährleisten
- Teile der Anwendungslogik kann wegen der mächtigen Datenmanipulationssprache (SQL)[20] in die Datenmodelle verlagert werden
- Datenmanipulationen werden nur über standardisierte Datenoperationen/Funktionen vorgenommen.
- Abfrageoptimierungen werden vom DBMS eigenständig durchgeführt durch Verwendung von Ausführungsplänen und Vorhersagen
- zentralisierter Zugriff und Manipulation auf operative Daten und somit Vermeidung von Redundanz bzw. kontrollierte Datenredundanz für verteilte Systeme
- Koordinierter Mehrbenutzerzugriff unter Berücksichtigung von Zugriffsberechtigungen und Authentifizierung damit der Eindruck einer exklusiven Nutzung des Systems besteht.
- Transaktionskontrolle (ACID)[21]: Somit immer konsistenter Zustand der Daten. Es kann kaum zu Datenverlust kommen. Eine Transaktion ist eine logische ununterbrechbare Arbeitseinheit der Anwendung an deren Beginn und nach dem Ende der Transaktion wieder ein konsistenter Zustand vorliegt.
- Definierte Zugriffsschnittstellen für alle Anwender/Anwendungen um auf einen gemeinsamen Datenbestand zuzugreifen.

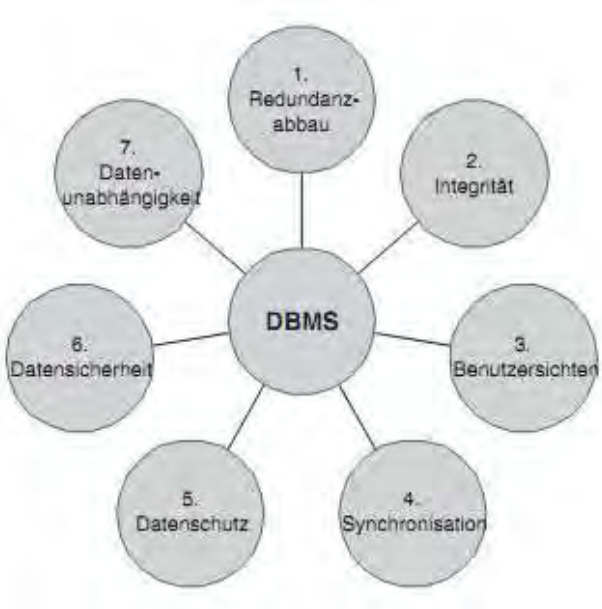


Abb. 8 Aufgaben/Vorteile eines Datenbank-Management-System[22]

[20] SQL: wird oft als 'Structured Query Language' bezeichnet, obwohl sie ein eigenständiger Name ist. SQL ist von ANSI standardisiert und wird von fast allen gängigen relationalen Datenbanksystemen unterstützt.

[21] Abkürzung für 'atomicity, consistency, isolation und durability' und steht für die Verlässlichkeit bei Datenmanipulationen

[22] [Cordts 02], p. 19

2.5 Der Datenbankentwurf

Um reale Objekte in einer Datenbank abzubilden, werden Modelle verwendet. Sie dienen dazu die Komplexität zu verringern und nur jene für die Aufgabenstellung wesentlichen Eigenschaften und Merkmale des zu beschreibenden Objektes zu berücksichtigen und somit die Realität möglichst aussagekräftig aber trotzdem einfach zu beschreiben.
Das **drei Ebenen Konzept** wird benutzt um Ausschnitte aus der realen Welt in einer Datenbank abzubilden:

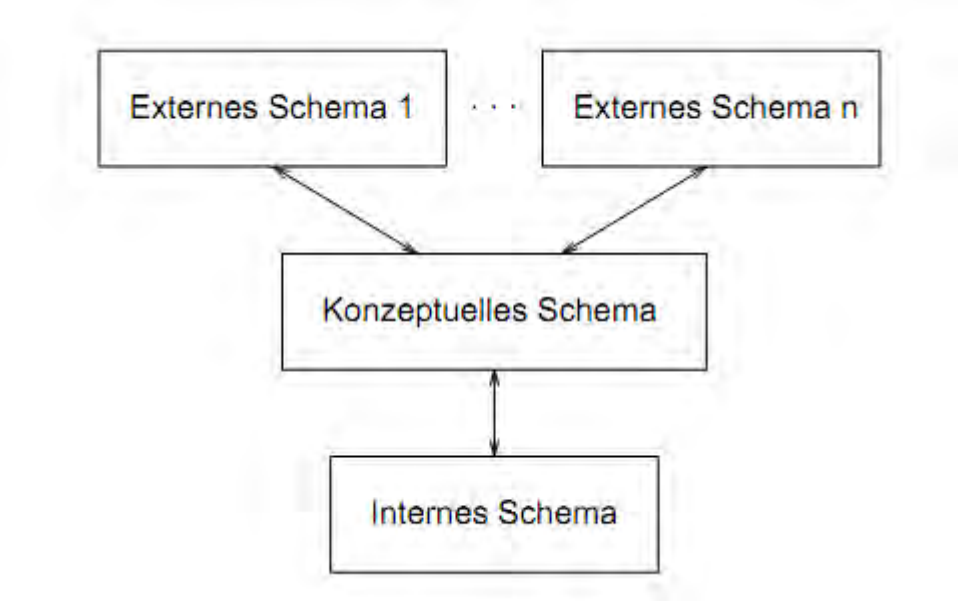


Abb. 9: Aufgaben/Vorteile eines Datenbank-Management-System[23]

Das **externe** Schema bildet den Teil eines Datenbankschemas ab, welchen der User von aussen sieht und mit ihm arbeitet. Es dient dazu, interne Systemdetails und die physische Speicherung der Daten vor dem Anwender zu verbergen.[24]
Das **interne** Schema beschreibt wiederum, wie die Daten physisch abgelegt und organisiert sind. Es ist sehr system- und rechnerabhängig und für den Anwender nur für Tuningmassnahmen interessant.[25]
Das **konzeptionelle** Schema gibt an, welche Datenobjekte vorhanden sind und wie sie untereinander in Beziehung stehen. Weil diese Ebene rechnerunabhängig ist, wird deshalb auch noch oft das in der Datenbank implementierte logische, konzeptionale Schema unterschieden.[26]

[23] [Saake 03], p. 4
[24] vgl. [Saake 03], p. 4
[25] ibid, p. 3
[26] ibid

Wichtig an dem drei-Ebenen-Konzept ist, dass z.B. Änderungen am konzeptionalen Schema keine Auswirkungen auf das Externe oder Interne Schema haben und dass Änderungen an der physischen (Internes-Schema) Ebene ebenfalls keine Auswirkungen auf das Externe-Schema nach sich ziehen. Das Konzeptionale-Schema wird mit einem allgemeinen systemunabhängigen Datenmodell beschrieben.

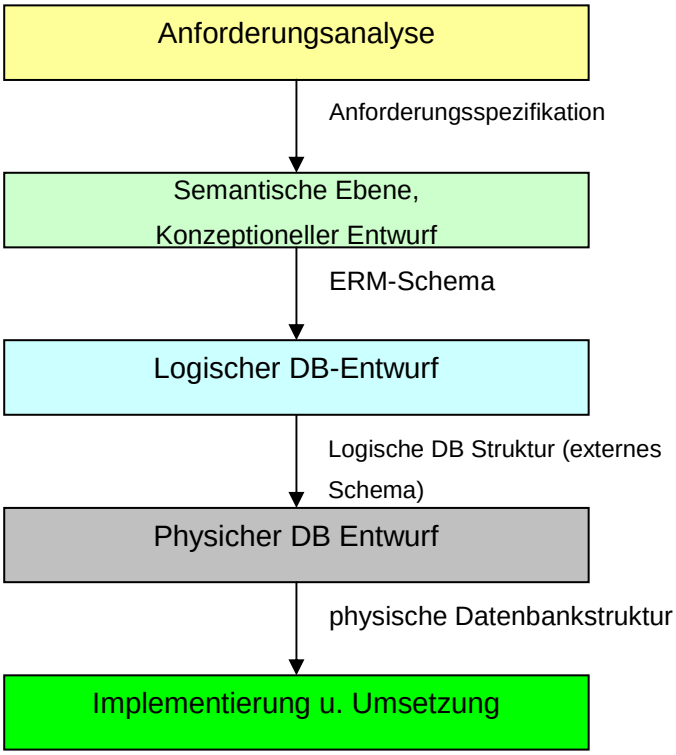


Abb. 10: vereinfachtes Vorgehensmodell zur Datenbankentwicklung

Der eigentliche Datenbankentwurf erfolgt dabei in verschiedenen Ablaufschritten und orientiert sich meistens an den Stufen von Vorgehensmodellen der Softwareentwicklung. Wie aus Abbildung 10 ersichtlich ist das drei-Ebenen-Konzept enthalten. Für den Entwurf des konzeptionalen Schemas werden allgemeine, systemunabhängige konzeptionelle Datenmodelle wie das Entity Relationship Modell[27] [28] verwendet. Aus diesem ERM wird dann das logische Datenbankmodell entwickelt.

[27] Entity bedeutet Objekt, aus dem Englischen und heisst übersetzt Gegenstands-Beziehungs-Modell

[28] Entity Relationship Modell, abgekürzt ERM

2.6 Datenbankmodelle[29]

Die beim Entwurf verwendeten und implementierten Datenbankmodelle können hinsichtlich ihrer

- Datenobjekte - definiert die syntaktische Struktur
- Datenstruktur - beschreibt die Zusammenhänge der Datenobjekte und ihrer
- Datenoperatoren - stellen die Manipulationsmethoden innerhalb des Datenmodells dar,

unterteilt werden.

In all diesen Modellen werden nur mehr konkrete Objekte und ihre Datenstruktur mit ihren Beziehungen zueinander betrachtet.

Bei den logischen Datenbankmodellen werden vor allem die hierarchischen und netzwerkartigen, sowie das relationale Datenbankmodell unterschieden. Wobei für diese Arbeit nur das logische relationale Datenbankmodel von Bedeutung ist:

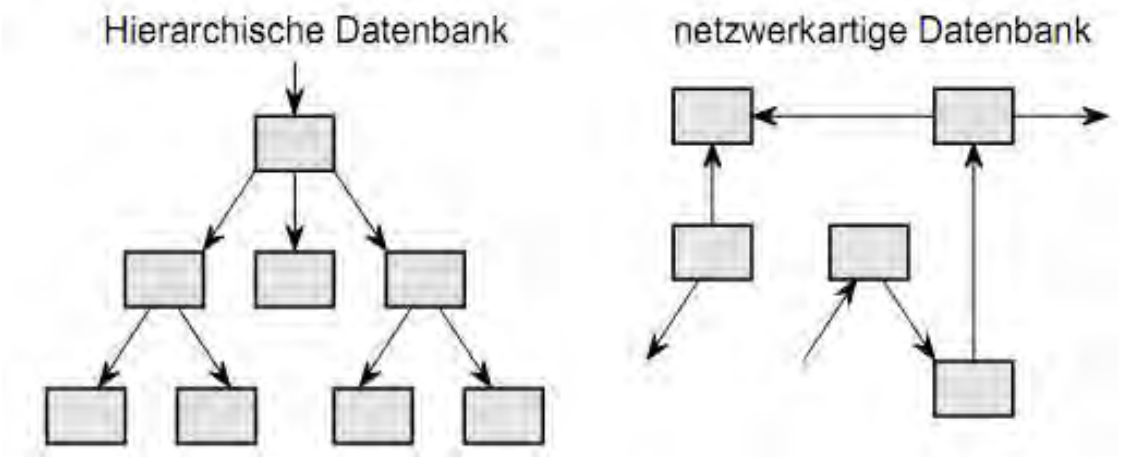


Abb. 11: hierarchische und netzwerkartige Datenbanken[30]

- **hierarchisches Datenmodell:** Es ist eines der frühesten Modelle und es ist abgeleitet aus der baumstrukturierten Dateisystemverwaltung. Es ist deshalb relativ vertraut, da viele reale Strukturen hierarchisch aufgebaut sind. Die Nachteile davon sind, dass m:n Beziehungen schwierig abzubilden sind und das Einfügen oder Löschen von Knoten ebenfalls schwierig ist.[31]

[29] Vgl. [Schicker 96], p. 56 ff

[30] [Schicker 96], p. 58

[31] [Saake 03], p. 7

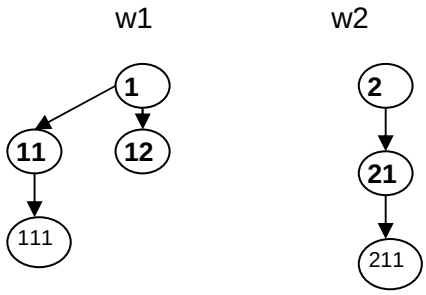


Abb. 12: hierarchische Datenmodell

Abbildung 12 zeigt ausgehend vom Knoten w1 einen Baum. Der Ursprungsknoten wird Wurzel genannt. Von ihm führen nur gerichtete Verbindungen zu anderen Knoten weg. w2 wird als Bogen bezeichnet, weil jeder Knoten jeweils nur durch eine gerichtete Verbindung verbunden ist. Mehrere Bäume und Bogen zusammen werden als Wald bezeichnet.

- **Netzwerk-Datenmodell**

  Im Netzwerk Datenmodell werden Daten von der gleichen Art als vernetzte Recordsets gespeichert. Ein Record kann mit vielen Records eines anderen Recordsets vernetzt sein. Eine Beziehung zwischen zwei Records darf stets nur in eine Richtung gehen, ähnlich der Baumstruktur, und heisst gerichteter Graph.

  Es hat zwei erwähnenswerte Vorteile:

  a) Die Beschreibung ist einfach und es gibt eindeutige Regeln für die Modellierung.

  b) Es sind sehr leistungsfähige DBMS vorhanden, die die Netzwerkschemen verarbeiten.

  Die wesentlichen Nachteile entstehen erst bei der praktischen Benutzung des Modells:

  a) um Daten zu manipulieren muss man durch das Netz navigieren,

  b) dafür muss man die Struktur des Modells genau kennen und das ist aus Anwendungssicht kompliziert.

  c) Bestimmte Wege ergeben sich nicht nur aus dem Modell sondern sind auch durch die interne Realisierung vorgegeben.

  d) Damit ist aber auch die physische Datenunabhängigkeit reduziert.

- **relationales Datenmodell**

  Objekte der realen Welt werden im Relationenmodell durch Tabellen dargestellt. Jede Tabelle setzt sich aus Zeilen und Spalten zusammen. Die Tabelle ist also eine Sammlung aller zugehörigen Zeilen. In einer Tabelle sind meist in ihrer Struktur gleichartige Objekte zusammengefasst. Die Spalten repräsentierten ein bestimmtes Merkmal mit seiner wertmässigen Ausprägung der entsprechenden realen Objekteigenschaft.

  Relationen zwischen den einzelnen Tupeln[32] sollen bestehende Beziehungen oder einen bestimmten Sachverhalt zwischen zwei Tabellen ausdrücken. Zusätzlich werden so genannte Schlüsselattribute vergeben, um zum einen die Zuordnung (Relation oder Beziehung) zwischen Objekten (Tabellen) darzustellen und zum anderen den Zugriff auf eine Tabelle eindeutig zu definieren.

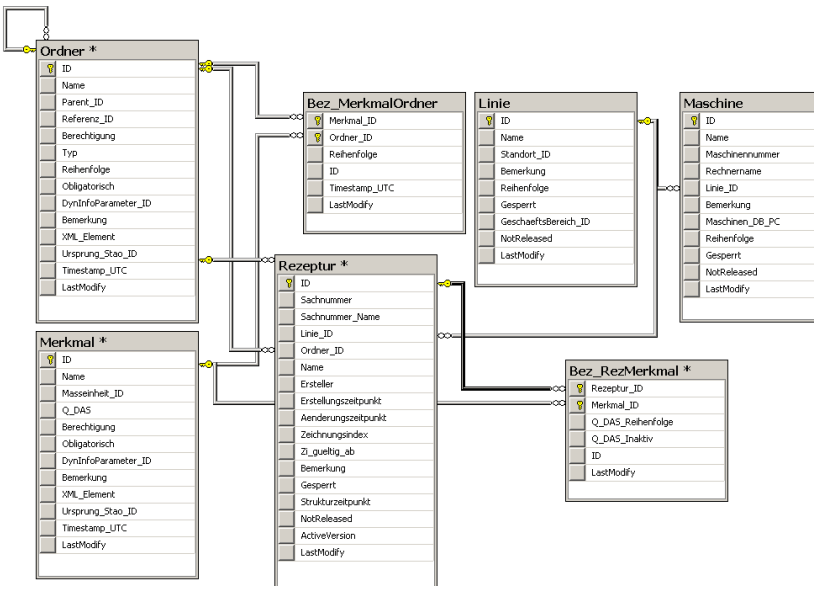


Abb. 13: relationales Datenmodell, ERD[33] von Teilen des RZNW-Datenbankschemas

2.7 Entity-Relationship-Modell[34]

Mit dem Entity-Relationship-Modell[35] wird ein abgrenzbarer, realer Gegenstand über ein ER Diagramm modellhaft beschrieben. Es wird nur beschrieben welche Merkmale die unterscheidbaren und selektierbaren Gegenstände/Objekte haben und wie sie miteinander in Beziehung stehen.

---

[32] Synonym für Datensatz oder noch mehr vereinfacht: Zeile

[33] ERD, abgekürzt für 'entity relationship diagram' aus dem Englischen übersetzt:,'Entitäten-Relationen-Diagramm'. Es ist eine grafische Darstellung der Tabellenstrukturen in einer DB. Abgekürzt für ER-Diagramm

[34] [Wieken 09], p. 224 ff.

[35] Abgekürzt: ER-Modell oder ERM

Dabei gelten folgende Bedingungen:

- Vollständigkeit
- Korrektheit
- Minimalität
- Lesbarkeit

Um so ein Modell zu entwerfen, geht man nach einem dreigliedrigen Abstraktionskonzept vor:

- Klassifikation - Assoziation:

  Sie fasst Objekte (Entities) mit gleichen Attributen zu einer neuen Entity-Menge oder Klasse zusammen. Für alle Objekte der neuen Entity-Menge oder Klasse gelten dann immer die gleichen Integritätsbedingungen und die gleichen Mengen-Operationen. In der relationalen Algebra ist das die Operation Mengenbildung
- Aggregation:

  Zusammenfassung möglicher unterschiedlicher Teilobjekte (Komponenten) zu einem neuen Objekt oder einem neuen Entity-Typ. In der relationalen Algebra entsprich das der Bildung von kartesischen Produkten.
- Generalisierung / Spezialisierung:

  Teilmengenbeziehungen zwischen Elementen verschiedener Klassen, z.B. ist der Entity-Typ 'Student der technischen Informatik' spezieller als die Obermenge 'Student'. In der relationalen Algebra wird das durch die Bildung von Potenzmengen (bzw. Teilmengen) ausgedrückt. Der Entity-Typ Student vererbt Eigenschaften an die Teilmenge 'Student der technischen Informatik' z.B. Matrikelnummer.

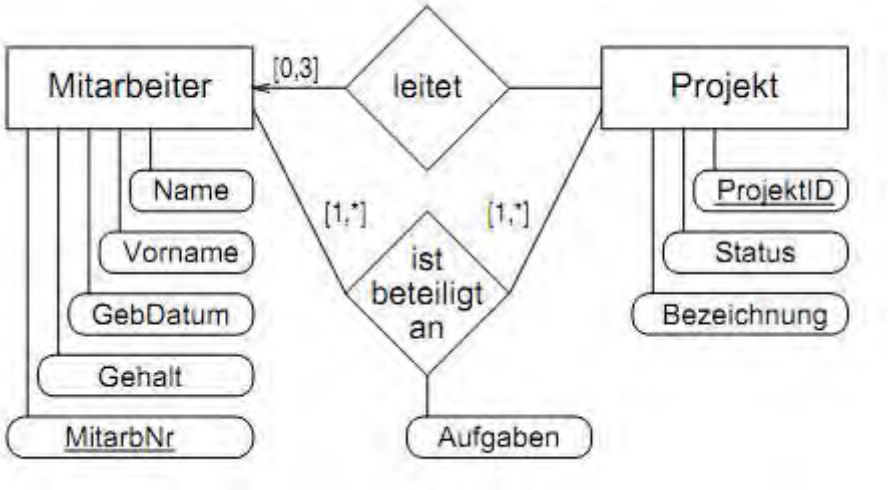


Abb. 14: Eine Beispiel-ER-Modellierung[36]

[36] [Saake 03], p. 31

Entity (Gegenstand):

Damit wird ein abgrenzbares Objekt aus der realen Welt beschrieben. Es kann ein realer Gegenstand oder auch nur eine gedankliche Abstraktion sein.

Beispiele: `Student X oder Prüfung Y`

Entity Typ:

Ein Entity-Typ ist die Zusammenfassung von gleichartigen Entities, welche die gleichen Eigenschaften oder Merkmale, aber unterschiedliche Ausprägungen (Werte) besitzen.

Beispiele: `Student oder Prüfung`

Die Merkmale eines Entity-Typs werden Attribute genannt.

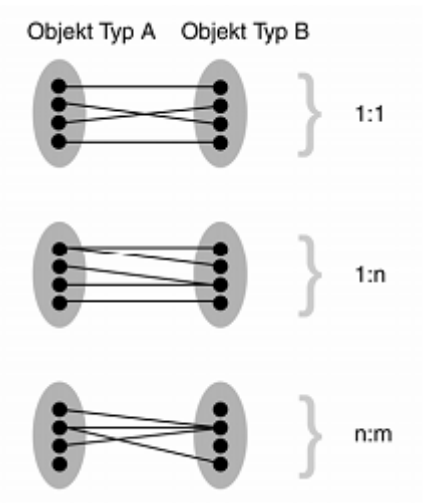
Beispiele: `Matrikelnummer eines Studenten, Prüfungsnummer`

Die konkrete Ausprägung eines Attributs eines bestimmten Entities wird Wert genannt.

Beispiel: `Matrikelnummer des Studenten X`

Beziehungen der Entity-Typen:

Dabei werden 3 Arten von Beziehungen unterschieden



1:1 Beziehung

Jedes Entity vom Typ i steht nur mit einem Entity vom Typ J in Beziehung

1:N Beziehung

Jedes Entity vom Typ i kann mit mehreren Entities vom Typ J in Beziehung stehen

M:N Beziehung

Mehrere Entities vom Typ i können mit mehreren Entities vom Typ J in Beziehung stehen

Abb. 15: relationale Beziehungen[37]

2.8 Relationales Datenmodell

Das relationale Datenmodell wurde von Dr. Edgar Frank Codd 1970 in seinem Artikel **'A Relational Model of Data for Large Shared Data Banks'**[38] vorgestellt um Datenunabhängig-

[37] [Skulschus 07], p. 26

keit zu erreichen. Das Modell ist aufgrund seiner Einfachheit und Flexibilität das bisher erfolgreichste Datenmodell. Es basiert auf dem mathematischen Konzept der Relation, bei dem Relationen (Mengen) in Tabellenform ausgedrückt werden. Die relationale Algebra ist die theoretische Grundlage und beschreibt, wie Daten manipuliert werden können.[39]

Die grundlegenden Operationen sind

- Projektion

  $$\pi_{\beta}(R) := \{t_{\beta} | t \in R\}$$

  Bei der Projektion werden einzelne Attribute (Spalten) aus der Attributmenge ausgeblendet. Sie ist als eine vertikale Selektion auf Spaltenebene zu verstehen
- Selektion

  $$\sigma_{\text{Ausdruck}}(R) := \{t | t \in R \wedge t \text{ erfüllt Ausdruck}\}$$

  Bei der Selektion wird über einen Vergleichsausdruck festgelegt, welche Tupel (Zeilen) in der Ergebnismenge berücksichtigt werden. Sie ist eine horizontale Selektion auf Zeilenebene.
- Kreuzprodukt (Kartesisches Produkt)

  $$R \times S := \{(a_1, a_2, ..., a_n, b_1, b_2, ..., b_m) | (a_1, a_2, ..., a_n) \in R \wedge (b_1, b_2, ..., b_m) \in S\}$$

  Das Resultat des kartesischen Produkts ist die Menge aller Kombinationen der Tupel aus R und S, d.h. jede Zeile der einen Tabelle wird mit jeder Zeile der anderen Tabelle kombiniert.
- Vereinigung

  $$R \cup S := \{t | t \in R \vee t \in S\}$$

  Bei der Vereinigung R mit S werden alle Tupel der Relation R mit allen Tupeln der Relation S zu einer einzigen Relation vereint.
- Umbenennung

  $$\rho_{[\text{neu} \leftarrow \text{alt}]}(R) := \{t' | t'(R - \text{alt}) = t(R - \text{alt}) \wedge t'(\text{neu}) = t(\text{alt})\}$$

  Durch diese Operation können Attribute (Spalten) und Relationen umbenannt werden, damit Operationen möglich sind, bei denen sonst doppelte Attributnamen aus den beteiligten Entity-Typen entstehen würden.
- Differenz

  $$R - S := R \backslash S := \{t | t \in R \wedge t \notin S\}$$

  Bei der Differenz der Relation R − S werden aus der ersten Relation R alle Tupel entfernt, die auch in der zweiten Relation S vorhanden sind

---

[38] [Codd 70], p. 26

[39] [Klein 05], p. 13 ff.

2.8.1 Begriffe der relationalen Datenbank, vom Kleinen zum Grossen

Eine relationale Datenbank besteht aus einzelnen Tabellen, den Relationen, in denen Datensätze als Zeile, dem Tupel abgespeichert sind. Die Spalten der Tabelle werden Attribute genannt. Die Werte, die in einer Spalte stehen können, sind einerseits abhängig vom Datentyp und andererseits von dem definierten Wertebereich.

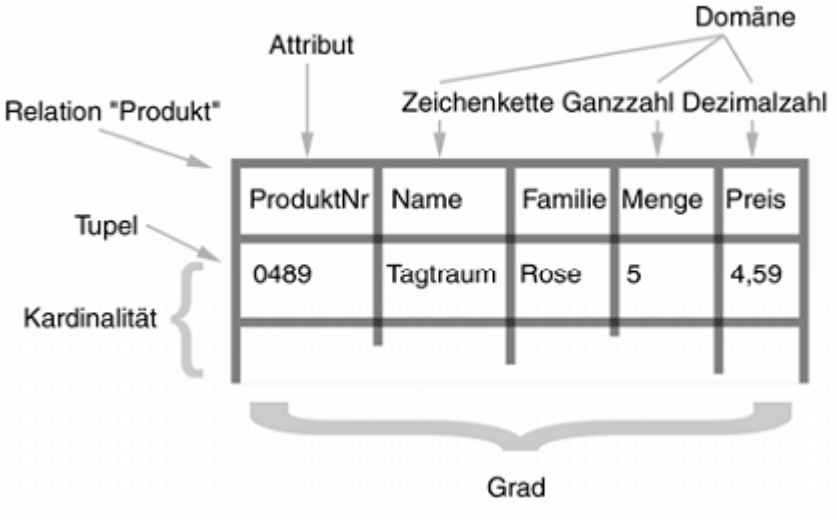


Abb. 16: Fachbegriffe für Tabellenbestandteile[40]

Domäne:

Eine Domäne besteht aus einem Namen D und einer Menge atomarer Werte und definiert den Wertebereich bzw. den Datentyp einer Spalte.

Beispiel:

```
Datum= {01.01.1970 – 12.31.2099}
Zahl = {0…99}
Buchstabe = {,A'…'Z','a'…'z'}
```

Tupel:

Ein Tupel t ist eine Liste mit n Werten t = <d1, d2, ..., dn>, wobei jeder Wert di ein Element der Domäne Di, oder NULL sein muss. Ein Tupel entspricht einem Datensatz der Relation oder einer Zeile der Tabelle.

Beispiel:

```
T1 = <Ford EMV, 10.10.1999, 2>
T2 = <Seat Lenkgetriebe, 02.04.2010, 122>
T3 = <VW Lenkeinheit, 17.02.2011, 523>
```

[40] [Skulschuss 07], p. 40

Attribut:

Ein Attribut A bezeichnet die Funktion, die eine Domäne Di in einem Relationenschema R ausübt. Jede Ausprägung oder Wert di des Attributs muss ein Element der Domäne oder NULL sein.

Beispiel:

> Bei einem Produktionsauftrag könnte die Domäne Zahl die Menge der produzierten Teile angeben. Die Domäne Buchstabe die Produktbezeichnung und die Domäne Datum das Produktionszeitpunkt.

Relationenschema:

Ein Relationenschema R, Schreibweise: R(A1, A2, ..., An), bezeichnet eine Menge von Attributen {A1, A2, ..., An}. Es legt die Anzahl der Attribute und ihre Domäne für eine Relation fest. Weiters muss ein Datensatz eindeutig identifizierbar sein. Dazu wählt man aus den Attributen einen Schlüssel, aus der sich nur auf den Datensatz und nicht die Position innerhalb der Relation bezieht. Mit dem Schema werden auch Integritätsbedingungen festgelegt.

Beispiel:

> Produktionsauftrag = { Produktname, Produktionszeitpunkt, Anzahl produzierter Teile}

Relation:

Eine Relation r ist eine Instanz (Ausprägung) des Relationenschemas R(A1, A2, ..., An). Sie ist eine Teilmenge des kartesischen Produkts (Kreuzprodukt) der beteiligten Domänen.

Relationales Datenbankschema:

Ein relationales Datenbankschema ist eine Menge von Relationenschemata S = {R1, ..., Rn} zusammen mit einer Menge von Integritätsbedingungen. Eine relationale Datenbankinstanz ist die Menge {r1, ..., rn}, wobei ri Instanz von Ri ist und alle Integritätsbedingungen erfüllt sind.

Relationale Datenbank:

Eine relationale Datenbank ist ein relationales Datenbankschema mit einer entsprechenden Datenbankinstanz.

Primärschlüssel[41]**:**

Wie schon erwähnt ist ein Attribut oder eine Kombination aus Attributen (z.B. in Beziehungs-Relationen) dann ein Schlüsselkandidat, wenn dadurch jeder Tupel in der Relation eindeutig bestimmbar ist. Eine Attributkombination ist dann minimal und ebenfalls ein Schlüsselkandidat, wenn keines der Attribute aus der Kombination gestrichen werden kann, ohne die Eindeutigkeit aufzugeben. Aus diesen Schlüsselkandidaten wird genau nur ein Schlüssel als Primärschlüssel der Relation ausgewählt. Über ihn kann nun jeder Datensatz in der Relation **eindeutig** identifiziert werden. Häufig ist das auch ein zusätzlich eingeführtes Attribut wie z.B. ein Attribut mit dem Datentyp 'Uniqueidentifier'[42] oder die Produktnummer usw. Der Wert eines Primärschlüssels darf über seine Lebensdauer nicht abgeändert werden.

Tip: Ist der Primärschlüssel ein zusammengesetzter Schlüssel, so sollte ein neuer künstliches Attribut eingeführt werden. (zusammengesetzte Schlüssel können nicht repliziert werden (unter MS SQL Server werden solche Relationen automatisch um ein UUID Attribut erweitert)).

Fremdschlüssel[43]**:**

Gerade wenn in Tabellen Beziehungen zu anderen Tabellen dargestellt werden möchten, wird meist der Primärschlüssel aus der Ursprungs-Tabelle als Attribut in der anderen Relation aufgenommen. Da er zur Identifizierung von Tupeln einer fremden Relation dient, wird er deshalb Fremdschlüssel genannt.

Strukturelle Integritätsbedingungen:

Eine Datenbank wird dann konsistent oder integer genannt, wenn die darin enthaltenen Daten fehlerfrei erfasst sind und auch fehlerfrei wiedergegeben werden können.

Eine Integritätsbedingung ist eine Einschränkung der möglichen Ausprägungen zu einem Datenbankschema. Die Datenintegrität ist dagegen nicht gegeben, wenn Anomalien, Mehrdeutigkeiten oder widersprüchliche Sachverhalte auftreten. Die wichtigste Integritätsbedingung in relationalen Datenbanken ist die Schlüsselbedingung.

Der **Primärschlüssel** gewährleistet die **Eindeutigkeitsbedingung**.

Fremdschlüssel verweisen auf andere Tupeln. Es ist sinnvoll, wenn jeder Wert eines Fremdschlüssels als Primärschlüssel in der referenzierten Relation enthalten ist. Ist das gegeben, nennt man das **Referentielle Integrität.**

---

[41] Primärschlüssel: es wird meistens die englische Bezeichnung verwendet: primary key.

[42] Korrekt ausgeschrieben: Universally Unique Identifier (UUID) ist ein Attribut-Wert der mit der SQL Funktion NEWID generiert wird und weltweit eindeutig ist. Er wird aus dem Zeitstempel und der Mac-Adresse der Netzwerkkarte generiert. Der Aufbau ist {8 Stellen}-{4 Stellen}-{4 Stellen}-{4 Stellen}-{12 Stellen}.

[43] Fremdschlüssel: es wird meistens die englische Bezeichnung verwendet: foreign key

Daneben wird noch die Wertebereichsbedingung unterschieden, die dafür verantwortlich ist, dass Attribute aus einer Relation nur Datenwerte aus der Domäne (Wertebereich) annehmen können.

Transaktionen ACID-Eigenschaften:

- **Atomicity**: Eine Transaktion (TA) wird entweder vollständig oder gar nicht ausgeführt.
- **Consistency**: Eine TA hinterlässt einen konsistenten DB-Zustand im Sinne der Integritätsbedingungen (sofern die DB vor der TA konsistent war); anderenfalls wird sie zurückgesetzt. Zwischenzustände während der TA Durchführung dürfen inkonsistent sein.
- **Isolation**: Keine Beeinflussung zwischen nebenläufigen (parallelen) TAs z.B. durch eine andere Anwendung. Jede einzelne TA läuft so, als wäre sie allein im System (logischer Einbenutzerbetrieb). Das Endergebnis mehrerer nebenläufiger TAs ist identisch mit der Hintereinanderausführung der Befehle. (Serialisierbarkeit → Isolation → logischer Einbenutzerbetrieb)
- **Durability**: Die Ergebnisse abgeschlossener TAs bleiben dauerhaft erhalten, auch nach schweren Systemfehlern

2.9 Vom ERM zum relationalen Datenmodell

Um ein semantisches, konzeptionales Schema wie dem Entity-Relationship-Modells, in ein relationales Modell überzuführen gibt es einige Bedingungen und Regeln einzuhalten. Mit den Regeln werden die Entities und Relationen[44] auf Tabellen, Attribute, Schlüssel und Integritätsbedingungen im relationalen Datenbankschema umgeformt.

2.9.1 Transformation und Umsetzung des ERM in ein Relationen Modell

Vereinfachte Regeln für die einzelnen Objekte im ERM:

Entitätsmengen:

- Jeder Entity-Typ im ERM entspricht einem Relationenschema und alle Attribute werden übernommen.
- Zerlege dabei zusammengesetzte Attribute in einzelne Attribute.
- Der Identifikationsschlüssel oder ein Schlüsselkandidat des Entity-Typs wird zum Primärschlüssel in der Relation.

[44] Achtung: im ERM wird unter Relation Beziehung zwischen den Entitäten verstanden, wo hingegen im relationen Modell mit den Relationen die Tabellen gemeint sind

Beziehungsmengen:

Eine Beziehung (Relationship-Typ) im ERM wird in einer Relation so abgebildet, dass jeweils die Schlüsselattribute aller an der Beziehung beteiligten Entity-Typen und die eigenen Attribute der Beziehung in der Relation enthalten sind.

Der Primärschlüssel der Beziehungsmenge ist meistens die Kombination der Fremdschlüssel der an der Beziehung beteiligten Entity-Typen oder es wird ein neuer künstlicher Schlüssel erstellt (Identity).

Für **1:1-** und **1:n-Beziehungen** kann dabei auch eine Einbettung der gesamten Relation in die entsprechende Entity-Relation erfolgen.

Eine **einfach-komplexe (1:n)** Beziehungsmenge kann ohne eine eigenständige Beziehungstabelle durch die beiden Relationen der zugeordneten Entitätsmengen ausgedrückt werden. Dazu wird in der Tabelle mit der einfachen Beziehung (Assoziation) der Identifikationsschlüssel der referenzierten Tabelle als Fremdschlüssel geführt.

komplex-komplexe Beziehungen, m:n Beziehungen

Jede m:n Beziehung entspricht einer eigenständigen Relation, bei der ebenfalls die Primärschlüssel der beteiligten Entity-Typen als Fremdschlüssel (Attribut) hinzugefügt werden. Der Primärschlüssel kann wieder der aus den Fremdschlüsseln zusammengesetzte Schlüsselkandidat sein.

Generalisation:

Jede Entitätsmenge einer Generalisationshierarchie entspricht einer eigenständigen Relation, wobei der Primärschlüssel der übergeordneten Tabelle auch Primärschlüssel der untergeordneten Tabellen wird.

Aggregation:

Bei einer Aggregation entsprechen sowohl die Entitätsmenge als auch die Beziehungsmenge je einer eigenständigen Relation, falls der Beziehungstyp komplex-komplex ist. Die Tabelle der Beziehungsmenge enthält in diesem Fall zweimal den Schlüssel aus der zugehörigen Entitätsmenge als zusammengesetzten Identifikationsschlüssel, mit entsprechenden Rollennamen. Im Falle einer einfach-komplexen Beziehung (hierarchische Struktur) kann die Entitätsmenge mit der Beziehungsmenge zu einer einzigen Tabelle kombiniert werden.

Unter Umständen können dabei mehr Relationen erstellt werden als notwendig. Aus diesem Grund sind auch noch andere Regeln anzuwenden wie z.B. die **Normalisierung**.[45]
Ein Relationsschema sollte einen Entity-Typ oder einen Beziehungstyp beschreiben. Das heisst eine Relation sollte Informationen beinhalten, die tatsächlich auch logisch zusammengehören. Wird dies nicht beachtet, kann es zu Anomalien kommen. Um die Anomalien mehr oder weniger vollständig zu vermeiden und Redundanzen bei den Attributen zu reduzieren, werden verschiedene Normalformen für relationale Datenbankschemas vorgeschlagen.

2.9.2 Normalisierung[46]

Das Ziel der Normalisierung von relationalen Datenbankschemata ist es, alle Abhängigkeiten auf Schlüsselabhängigkeiten zurückzuführen. Hierzu müssen alle Relationen Schemen, in denen dies nicht gilt, aufgeteilt werden.

1) Die **erste Normalform 1NF** fordert die Atomarität von Attributwerten, d.h. es dürfen keine zusammengesetzten Attributwerte vorkommen.

Tabelle in 1NF	
1001	w1
1001	w3
…	….

Tabelle 1: 1NF

Tabelle nicht in 1NF	
1001	w1,w3
…	….

Tabelle 2: nicht in 1NF

Die Tabelle 1 befindet sich in 1NF. Werden diese zwei Reihen so geschrieben, dass die Werte des Attributs 2 zusammengesetzt würden (siehe Tabelle 2), wäre sie nicht mehr in 1NF sein.

[45] Normalisierung ist im Datenbankentwurf ein sehr wichtiges Thema, aber kann in dieser Arbeit nur erwähnt werden

[46] [Matt 09], p. 83 ff.

2) Die **zweite Normalform 2NF** verbietet zusätzlich zur 1NF Abhängigkeiten vom Schlüsselteilen. Alle Spalten, die nicht zum eindeutigen Schlüssel einer Tabelle gehören, müssen von dem gesamten Schlüssel abhängig sein.

student_nr	prof_nr	thema	beginn	studiengang
1001	P1	DB Replikation	01.02.2010	Technische Informatik
1001	P1	Verteilte DB	05.04.2010	Technische Informatik
1003	P2	Stabilisierung	16.02.2010	Wirtschaft

Tabelle 3: Studenten-Projektarbeiten nicht in 2NF

Den Primärschlüssel der Tabelle 3 bildet die Kombination der Spalten 'student_nr' und 'prof_nr'. Die Spalte 'studiengang' ist nicht voll funktional vom kompletten Primärschlüssel, sondern schon von einem Teil (studnet_nr) abhängig. Die oben dargestellte Tabelle befindet sich daher nicht in der 2NF.

3) Die **dritte Normalform 3NF** verbietet zusätzlich zur **2NF** transitive[47] Abhängigkeiten. Ein bekanntes Beispiel für die Verletzung der 3NF ist die folgende transitive Abhängigkeit:

`Firma → Adresse → Postleitbezirk`

Ein Attribut darf nicht funktional von einem anderen Attribut abhängig sein. In diesem Beispiel sind aber die Attribute Adresse und Postleitbezirk voneinander direkt abhängig. Da keiner der Attribute zum Primärschlüssel gehört wäre so eine Tabelle nicht in 3NF.

Alle Spalten einer Tabelle müssen vom Primärschlüssel der Tabelle und nicht bereits von anderen Spalten abhängig sein.

Ein Relationenschema in 3NF ist auch gleichzeitig in 2NF.

Relationen mit nur einem Nichtschlüsselattribut sind auch automatisch in der 3NF.

Es gibt noch weitere Abhängigkeitsklassen, die aber für die Praxis weniger relevant sind.
Im Anhang sind die **Coddsche**(n) **Regeln** aufgeführt mit denen ein DBMS überprüft werden kann, ob es das relationale Datenbankmodell unterstützt.

[47] Transitive Erklärung: es bestehe eine Relation x zwischen R und S. Außerdem besteht eine Relation der gleichen Art x zwischen S und T. Ist die Beziehungsart x transitiv, so besteht somit implizit auch zwischen R und T eine Beziehung x.

2.10 Datenbank Abfragesprache SQL

Die 'Structured Query Language' ist eine relationale Datenbanksprache und wird heute de facto in allen relationalen DBMS verwendet. Sie ist eine logische, mengenorientierte, nicht-prozedurale Sprache der 4. Generation. SQL ist standardisiert, plattformunabhängig und kann zur Beschreibung, Abfrage und Manipulation von Daten in relationalen Datenbanken genutzt werden. 'Nichtprozedural' bedeutet, der User gibt an, mit welchen Daten er arbeiten will und nicht wie der Zugriff auf diese zu erfolgen hat

Die Sprache ist aufgeteilt in:

- DDL (**data definition language**): Die Datendefinitionssprache dient zur logischen Datenbankbeschreibung. Über ihre Befehle können Datenstrukturen generiert, verändert und gelöscht werden. Sie dient weiters zur Verwaltung der Meta-Daten zur Beschreibung des Schemas wie Integritätsregeln usw.
- DML (**data manipulation language**): Mit der Datenmanipulationssprache werden Abfragen und Manipulationen an den Datenobjekten durchgeführt. Die Anfragesprachen sind deklarativ.
- DCL (**data control language**): Diese Befehle dienen der Vergabe von Zugriffsrechten in einer Datenbank, die von mehreren Personen genutzt wird

SQL wurde von IBM ab 1974 als Sprache für das relationale DBMS System R entwickelt.
Es gibt unter anderen folgende SQL Implementierungsversionen:

Jahr	Bezeichnung
1985	SQL1
1992	SQL92 oder SQL2
1999	SQL3
2003	SQL:2003 und 2008 SQL:2008 ISO/IEC 9075:2008

Tabelle 4: ausgewählte SQL Implementierungsversionen

Viele Hersteller wie Oracle oder Microsoft haben in ihren Systemen Erweiterungen von SQL implementiert. Andererseits ist meist nur der so genannte Entry-Level von SQL implementiert, somit ist es notwendig, jedes DBMS und die Abfragesprache im Detail zu kennen.

2.11 Verteilte Datenbanken.

Als Verteilte Datenbanken werden räumlich getrennte, autonome Datenbanken bezeichnet, die zu einem Datenbankverbund zusammengeschlossen sind. Eine verteilte Datenbank heisst dann replizierte Datenbank, wenn Kopien von Datenobjekten und ihrer Werte in mehreren Datenbanken existieren.

Die Vorteile von verteilten Datenbanken sind, dass eine

- echte **Nebenläufigkeit** erreicht wird, dass also Abfragen parallel und nicht nur quasi parallel abgearbeitet werden.
- eine **Skalierbarkeit** der vorhandenen Ressourcen wird erst möglich, da die Hardware so zusammengefügt werden kann, dass genau soviel Systemleistung für die Lösung der Aufgabe zur Verfügung steht.
- Mit verteilten Datenbanken kann neben der Erhöhung der Zugriffsgeschwindigkeit auch die **Ausfallssicherheit** erhöht werden, da mehrere Rechner die gleichen Daten bereitstellen oder Aufgaben dynamisch übernehmen können, ohne dass der User viel davon mitbekommt.

Somit sind die Hauptgründe für verteilte Datenbanken:

- Aufgabenlastverteilung durch Kooperation und paralleler Taskausführung
- Erhöhung der Ausfallsicherheit durch Redundanz von angebotenen Funktionen (Failover Cluster)
- somit geringere Fehleranfälligkeit
- einfachere Skalierbarkeit
- einfachere Administrierung, da einzelne Systeme abgeschaltet werden können, ohne das Gesamtsystem erheblich zu beeinflussen
- Standardhardware ist oft günstiger als ein Supercomputer.

2.12 Zusammenfassung

Mit relationalen Datenbanksystemen lassen sich grosse Mengen an persistenten Daten effizient, effektiv und hoch dynamisch verwalten. Vorhandene Daten können in immer neuen Konstellationen aufgrund der Anforderungen aggregiert und aufbereitet werden. Auch in absehbarer Zukunft werden relationale Datenbanken die Basis für alle Arten von Informationssystemen und Datenverwaltungssystemen sein.

3 Replikation in verteilten Datenbanken

In diesem Kapitel werden die verschiedenen Konzepte von Replikationsverfahren beschrieben sowie deren Vor-/Nachteile und mögliche Anwendungsfälle aufgezeigt.

3.1 Abgrenzung: Verteilte Systeme und Parallelität

Ein verteiltes System besteht aus autonomen Subsystemen, die oft auch weit entfernt voneinander stehen, bei denen es aber vor allem um die koordinierte Zusammenarbeit geht, um gemeinsam Aufgaben abzuarbeiten.

Das für verteilte Systeme charakteristische Kernproblem ist der Mangel an zentralisiertem Wissen.

Ein paralleles System besteht aus einer Vielzahl gleichartiger Subsysteme (Komponenten), die lokal zueinander angeordnet sind und nur einen geringen Grad an Autonomie aufweisen.

Charakteristisch ist eine enge und hochgradig parallele Bearbeitung eines Benutzerauftrags.

In dieser Arbeit liegt der Hauptfokus auf räumlich getrennt verteilte Datenbanken zum Zwecke der lokalen Abfrageautonomie aufgrund von begrenzter Netzwerkbandbreite oder häufigem Netzwerkunterbruch zu einer zentralen Datenbank.

3.2 Anwendungsszenarien von Datenbankverbunden

Um zu verstehen, was unter dem Begriff Replikation gemeint ist, sind nachstehend verschiedene Anwendungsfälle aufgeführt, wie Daten in einer Systemlandschaft vorgehalten werden können.[48]

Zentrale Datenhaltung:

Anwender greifen von räumlich getrennten Standorten aus auf eine zentrale Datenbank zu. Die Konsistenzsicherheit ist sehr hoch, da nur an einer Datenbasis operiert wird. Allerdings ist der Zugriff stark vom Netzwerk abhängig und in vielen Anwendungsfällen (mobile Aussendienstmitarbeiter, Netzwerk-Bandbreite) gar nicht permanent gegeben.

[48] [Tiem 09] p. 85 f

Verteilte Datenhaltung:

Daten werden in räumlich voneinander getrennten Datenbanken gehalten. Solange keine Daten von anderen Standorten benötigt werden, ist das auch kein Problem, allerdings ist es dann nicht möglich, z.B. von einer Konzernzentrale aus auf die Daten der Niederlassung vor Ort zuzugreifen.

Datenreplikation:

Auf verschiedenen Standortdatenbanken liegen Kopien derselben Daten und Teile davon werden in definierten Zeitabständen miteinander repliziert. Somit können sonst standortübergreifende Abfragen auf lokal vorgehaltene Daten erfolgen. Je nach Anwendungsfall nimmt man in Kauf, dass auch zeitlich begrenzte Daten-Inkonsistenzen auftreten können. Z.B. ist der Lagerbestand eines schnell umschlagenden Artikels aufgrund seiner dynamischen Veränderung kritischer, als z.B. die Anschrift eines Kunden die sich verhältnismässig weniger oft ändert und daher mehr statisch ist.

3.3 Replikation

3.3.1 Definition Replikation[49]

Unter Replikation versteht man allgemein, die mehrfache Speicherung gleicher Daten auf autonomen Datenbanken und diese dann auch fortlaufend konsistent zu halten. Der Hauptanwendungsfall besteht darin, Teile der Datenbestände von räumlich getrennten Datenbanken miteinander abzugleichen, zu replizieren. Dadurch kann die Zugriffsgeschwindigkeit erhöht werden, bzw. ein Zugriff (Verfügbarkeit) auf die Daten erst ermöglicht werden. Der Ausfall von einzelnen Replikaten beeinflusst die Verfügbarkeit des ganzen Systems je nach Architektur gering bis gar nicht.

[49] [Beut 96] p. 203 ff.

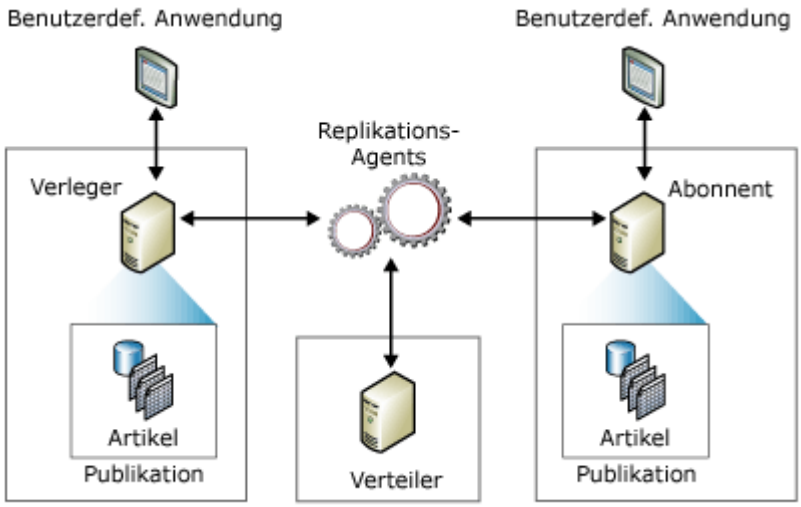


Abb. 17: Das Replikationsablauf [50]

Microsoft schreibt dazu: „Bei der Replikation handelt es sich um eine Reihe von Technologien zum Kopieren und Verteilen von Daten und Datenbankobjekten aus einer Datenbank in eine andere und das anschließende Synchronisieren zwischen den Datenbanken, um die Konsistenz der Daten sicherzustellen. Mithilfe von Replikation können […] Daten an verschiedene Standorte, an Remotebenutzer oder mobile Benutzer über lokale Netzwerke und WANs (Wide Area Network), über DFÜ-Verbindungen, drahtlose Verbindungen oder über das Internet verteilt“[51] werden.

3.3.2 Vorteile der Replikation

Mittels Replikation sollen folgende Vorteile erreicht werden[52]

- Um die **Ausfallssicherheit** von Datenbanken zu erhöhen, z.B. Kontoführungsdatenbanken in der Finanzwelt, Reservierungs- und Bestellsysteme.
- Für Datenbank Anwendungen auf **mobilen Clients,** die meistens **offline** sind (disconnected computing).
- Die **Verteilung der Abfrage-/Leselast** auf mehrere Datenbanken wie z.B. bei grossen Internetshops wie Amazon.com, wo in den meisten Fällen nur Lesezugriffe stattfinden und diese verteilt werden können. Schreibzugriffe bei Bestellungen werden oft auf dedizierte Server weitergeleitet.

[50] /MSRep 11/
[51] [Repl 11]
[52] [Beut 96] p. 203 ff

- **Reduzierung der Netzwerklast** zu einer zentralen Datenbank bei zu langsamen Netzwerk-Verbindungen.
- **Horizontale und vertikale Fragmentierung** der Daten aus einer Datenbank zu einer replizierten Datenbank, damit unternehmenskritische Daten erst gar nicht verteilt werden.
- Der Benutzer sollte keinen Unterschied zwischen dem Zugriff auf replizierte und nicht-replizierte Daten bemerken[53], deshalb muss die **Replikation transparent** für den Benutzer sein und die wechselseitige Konsistenz der Replikate gewährleistet sein. Die Benutzer erwarten, dass auch verzahnte Transaktionsausführungen seriell, also hintereinander ausgeführt werden.

Beispiel für die Wichtigkeit der Daten Konsistenz:

Eine Grundforderung von Datenbanken ist es, ihre Daten konsistent zu halten. Nicht nur physisch, sondern auch logisch. Man stelle sich ein Reiseunternehmen vor, welches nur über ein bestimmtes Kontingent an Plätzen pro Reiseangebot verfügt. Unterhält das Reiseunternehmen mehrere Reiseagenturen an verschiedenen Orten, so ist es für den Reiseagenten vor Ort wichtig, zu jedem Zeitpunkt genau zu wissen, wie viele Plätze noch frei sind. Dazu müssen die Daten auch aktuell sein. Doch nicht immer ist es notwendig, dass alle Daten unmittelbar synchron sind, sondern es reicht aus, wenn die Daten eine definierte Aktualität[54] haben.

3.3.3 Anwendungsszenarien für die Datenbankreplikation

Weitere Beispiele, in denen die Replikation zum Einsatz kommt:

Datenkonsolidierung: Konsolidierte Daten für die Verkaufszentrale bei Verkaufshäuserketten: Arbeiten die lokalen Verkaufsniederlassungen relativ autonom mit ihrer eigenen Datenbasis, so ist es für die Verkaufszentrale z.B. am Ende der Woche interessant über die Verkaufszahlen aller Niederlassungen zu verfügen, um z.B. die termingerechte Nachschubver-

[53] 1-Kopie-Äquivalenz: = Replikationstransparenz, es kann mehrere Replikate/Kopien innerhalb eines verteilten Systems geben. Änderungen müssen an allen Kopien vorgenommen werden. Somit sind alle Kopien auch zueinander konsistent. Englisch: 1-copy equivalence:

[54] auch Latenzeit: Sie ist jene Zeit, die verstreicht, bis alle Replikate wieder konsistent sind. [Hor 00]

sorgung der Niederlassungen zu gewährleisten. Die benötigten Daten werden dazu von den Niederlassungen zur Zentrale repliziert.[55]

Datenverteilung: Umgekehrt kann es notwendig sein, geänderte Artikelbezeichnungen oder zentral vorgegebene Daten auf die Standort Datenbanken der Niederlassungen zu replizieren, um die Abfragelast der Datenbank in der Zentrale zu minimieren.[56]

Replikation operativer Daten in Systemen zur Entscheidungsunterstützung: Sollen z.B. Berichte aufbereitet werden, so reichen als Basis meistens historische Daten. Dazu werden Daten aus operativen Datenbanken (OLTP Systemen, Online Transaction Processing) über z.B. synchrone Replikationsverfahren in eigene Datenbanken repliziert, auf denen dann oft Anwendungen laufen um Kennzahlen über OLAP (Online Analytical Processing) zu berechnen die unter Umständen sehr performanzlastig sind, je nachdem, wie die Daten aggregiert werden müssen.[57]

3.4 Ablauf der Replikation

Wie man sieht, bedeutet die Replikation vor allem nicht die exakte Kopie aller Daten zu jedem Zeitpunkt, sondern meist die Konsistenzhaltung einer Teilmenge der Daten zu einem vom Anwendungsfall vorgegebenen Zeitpunkt. Replikation ist die bewusste Einführung von Datenredundanz, wohingegen die Aufgabe der Synchronisation ist, Änderungen an diesen redundanten Daten zu protokollieren und diese an den replizierten Daten ebenfalls durchzuführen. Die Synchronisation ist für die Konsistenz der Daten auf verteilten Datenbanken durch verschiedene Synchronisationsverfahren zuständig. Der Synchronisierungsprozess besteht aus mindestens 2 Phasen: Der Reintegration und der Rückübertragung. Die Reintegration bedeutet den Abgleich der lokal geänderten Daten auf den Replikationsmaster[58]. Falls Änderungen an dem gleichen replizierten Datensatz und am Replikationsmaster Datensatz vorhanden sind, muss eine Konfliktauflösung implementiert sein. Bei der Rückübertragung werden aktualisierte Daten im Replikationsmaster auf die replizierten Daten beim Replikat (Änderungsauslösung) übertragen.

[55] Vgl. [Hor 00], p. 1

[56] ibid

[57] Vgl. [Hor 00], p. 1

[58] Auch Publisher genannt

3.5 Zielkonflikte

Aus der Art der Replikation und der Synchronisierung lassen sich drei zueinander entgegenstehende Ziele ausmachen:

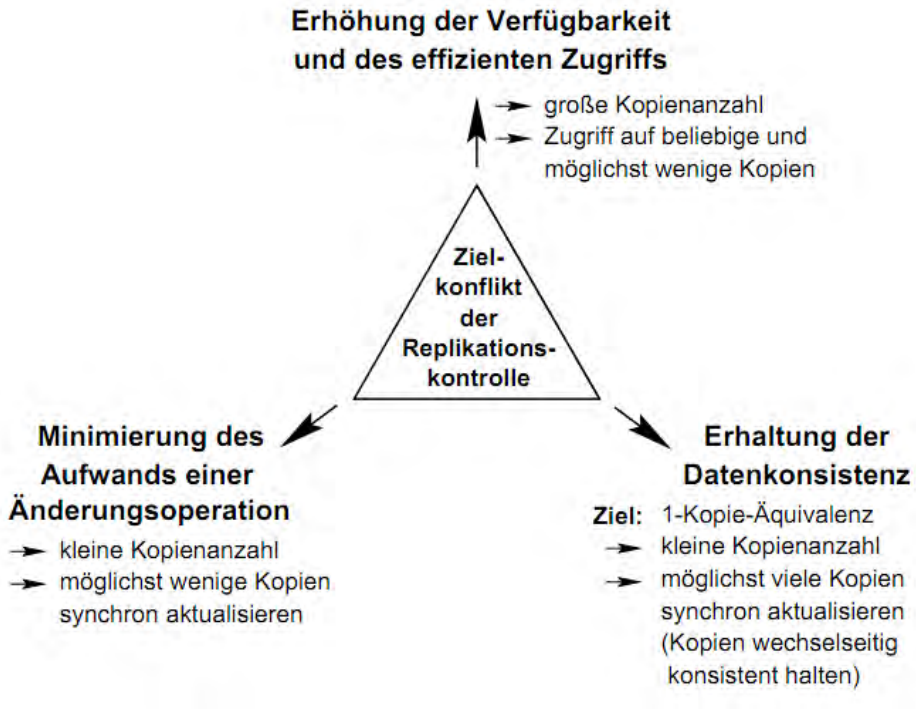


Abb. 18: Zielkonflikt der Replikationskontrolle[59]

Der Zielkonflikt besteht aus den Hauptforderungen nach Erhaltung der Datenkonsistenz, der Verfügbarkeit und der Performanz durch geringere zusätzliche Änderungsoperationen an den Daten. Die Datenkonsistenz ist umso einfacher zu erhalten, umso weniger Replikate eines Datenelements im Falle seiner Änderung ebenfalls geändert werden müssen. Die Performanz kann erhöht werden, wenn durch mehr Replikate der Lesezugriff verteilt werden kann und somit kürzer wird oder wenn der Lesezugriff auf ein netzwerktopologisch näheres Replikat erfolgt und somit die Antwortzeit kürzer ist. Andererseits steigt mit der Anzahl der Replikate auch der Aufwand bei Änderungen von Daten, diese Änderungen bei jedem Replikat nachzuziehen. Die Verfügbarkeit steigt wiederum mit der Fehlertoleranz, umso mehr Replikate vorhanden sind, umso unkritischer ist es für das Gesamtsystem, wenn ein Knoten ausfällt.

[59] [Beut 96] p. 204

3.6 Replikationsstrategien/ Replikationseinteilung

Da dieser Zielkonflikt besteht, muss für jeden Anwendungsfall individuell herausgefunden werden, wo das Optimum zu liegen kommt. Aus diesem Grunde wurden unterschiedliche Replikationsverfahren entwickelt, um Datenänderungen weiterhin zu synchronisieren und zu propagieren. Die strikte Transaktionskonsistenz von zentralen Datenbanksystemen wird für Replikate in der Regel aufgeweicht und durch die Konvergenz ersetzt, die für die Anforderungen an die Replikation wie Verteilung der Leselast und hoher und lokaler Verfügbarkeit der Daten besser geeignet ist.

Diese Replikationsverfahren können in **konsistenzerhaltende Verfahren** (strenge Konsistenz) und **verfügbarkeitserhaltende Verfahren** (schwache Konsistenz) unterteilt werden. In der Praxis haben sich die Verfahren mit schwacher Konsistenz mehr durchgesetzt, dabei werden die Replikate innerhalb der aktuellen Transaktion nicht mitberücksichtigt, sondern es reicht aus, dass alle Replikate nach einer gewissen änderungsfreien Zeit alle den gleichen Endzustand erreichen (konvergieren).[6061]

Die strenge Konsistenz wird bei replizierten Datenbanken auch als
1-Kopien-Serialisierbarkeit bezeichnet und wird definiert als:
„Ein Schedule S einer replizierten DB heißt 1-Kopien-serialisierbar, wenn es mindestens eine serielle Ausführung der TA dieses Schedule auf einer replikatfreien DB gibt, der die gleichen Ausgaben sowie den gleichen DB-Zustand erzeugt wie S auf der replizierten DB.“[62]

Es bedeutet, alle physisch getrennten Replikate werden als eine einzige logische Kopie betrachtet und die Ausführung von den darauf laufenden parallelen Transaktionen so verwaltet, also ob sie auf einem System ausgeführt würden.

Die Konsistenzanforderungen lassen sich mit zwei Parametern beschreiben, nach denen die Replikationsverfahren unterteilt werden können.[63]

1) Wo werden die Aktualisierungen ausgeführt, also an welchem Replikat?
2) Wann werden die Aktualisierungen zwischen den Replikaten vorgenommen?

[60] Vgl.[Beut 96] p. 204
[61] Vgl. [Mutsch 04], p. 88
[62] [Beut 96] p. 204
[63] [Gray 96], p. 172 ff.

Propagation **Vs Ownership**	Lazy (asynchron, optimistic)	Eager (synchron)
Group (Symmetrisch)	Änderungen von Daten bei jedem Replikat möglich (Update Everywhere). Bidirektionale Replikation. Aufgrund der zeitversetzten propagieren der Änderungen führt es zu schweren Inkonsistenzen, daher. gleichrangige Replikat Konfliktauflösungsmechanismen erforderlich. Merge Replikation N Updater, N^2 Transaktionen	Verteilte Transaktion notwendig. Keine Inkonsistenzen möglich. Strenge Konsistenz. Nur über 2PC realisierbar. Einziger Vorteil ist, Lesen überall autonom N Updater, N verteilte TA
Master Asymmetrisch	Updates nur an einem Ort, Master-Slave (Publisher, Subscriber) Topologie. Schwache Inkonsistenzen. Unterschiedliche Datenstände kurzzeitig vorhanden. Unidirektionale Replikation. Primary Copy Ansatz 1 Updater, N verteilte TA	Verteilte TA notwendig. Keine Inkonsistenzen. Datenbankspiegelungen mit Replikation Hot Standby Server 1 Updater, 1 verteilte TA

Abb. 19: Klassifikation der Replikationsverfahren[64]

Bei der Frage ‚wo werden die Aktualisierungen ausgeführt?', gibt es die Unterteilung in **symmetrische** und **asymmetrische** Verfahren. Bei der symmetrischen Konfiguration (Group) können Datenänderungen in allen Replikaten (Update Everywhere) durchgeführt werden und jedes Replikat erhält dann die Änderungen. Bei der asymmetrischen Konfiguration erfolgen Updates nur an einem Knoten (Master) während an allen anderen immer nur gelesen wird. Änderungen an einem Slave-Knoten leitet er zu der Primärkopie am Master um und der verteilt die Änderung an die Replikate weiter. Bei dem Unterscheidungsmerkmal nach dem Zeitpunk der Aktualisierungen wird zwischen **asynchron** (lazy) und **synchroner** (eager) Replikation unterschieden. In einer synchronen Aktualisierung werden alle Replikate gleichzeitig durch eine Änderungstransaktion aktualisiert und es kann zu keinen Dateninkonsistenzen oder Replikationskonflikten kommen. Durch die gemeinsame Transaktion kann es

[64] In Anlehnung an [Gray 96], p. 173 (kein vgl.)

zu erheblich lang andauernden Sperren an allen Replikaten kommen, da die Transaktion erst dann abgeschlossen wird, wenn sie an allen Replikaten auch durchführbar ist. In der Praxis wird dies über ein Zweiphasen Synchronisationsprotokoll[65] realisiert. Es müssen aber alle Replikate zum Transaktionsdurchführungszeitpunkt verbunden sein. Bei einer asynchronen Aktualisierung werden die Replikate zeitlich verzögert aktualisiert, somit ist die Transaktionszeit der Ursprungs Änderung kurz, weil erst nach deren Commit[66] am Master Replikat die Replizierung auf die anderen Replikate stattfindet. Das DBMS, in welchem die Änderung ihren Ursprung nahm, verhält sich wie ein normales DBMS bezüglich der Transaktionsverarbeitung und der Sperren. In einem solchen Szenario kommt es zeitweise zu Inkonsistenzen, die zu einem späteren Zeitpunkt durch Konfliktauflösungsverfahren behoben werden müssen. Der Vorteil aber ist, dass durch die Vermeidung von langen Sperren, die Daten hoch verfügbar bleiben.

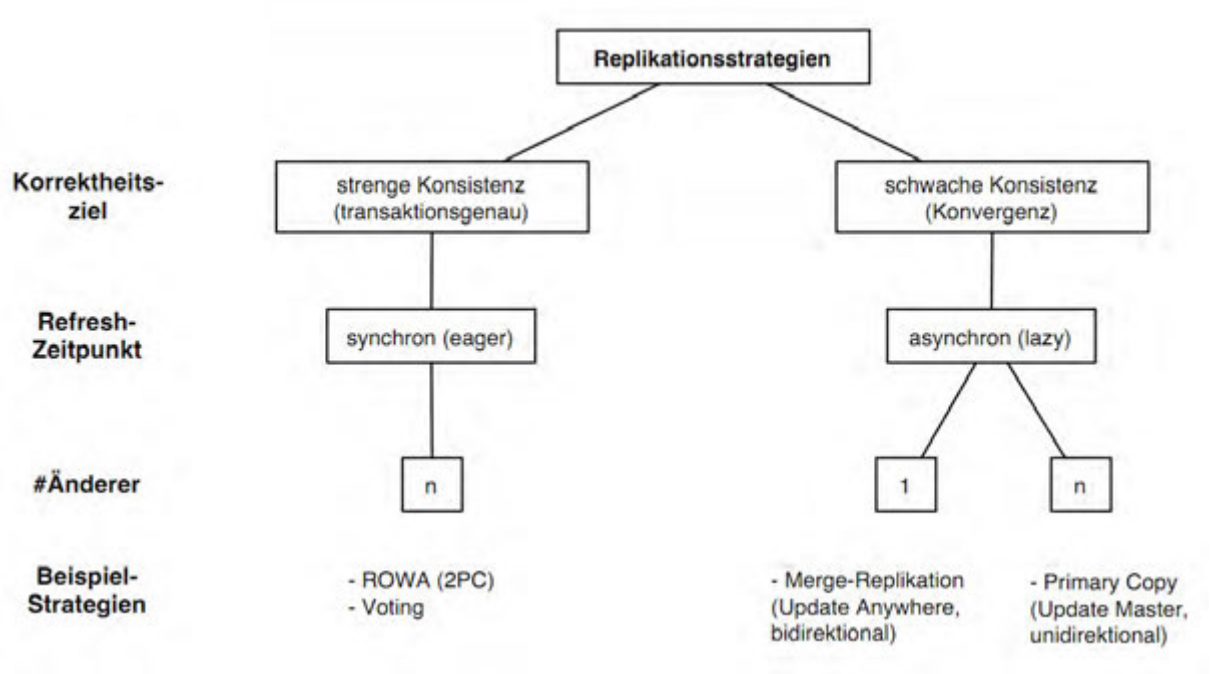


Abb. 20: Klassifikation der Replikationsstrategien [67]

[65] 2PC: in der ersten Phase, der Commit-request Phase, holt der TA Koordinator die Zustimmung oder Ablehnung aller beteiligten Prozesse ein. Danach entscheidet der TA Koordinator bei 100% Zustimmung auf Commit oder Rollback und benachrichtigt in der zweiten Phase, der commit phase alle beteiligten Prozesse über das Ergebnis.

[66] Ein Commit ist die Festschreibung der Daten mit dem Transaktionsabschluss

[67] [Mutsch 04], p. 9

3.7 Replikationskonflikt

Vor allem bei der symmetrischen, asynchronen Replikation kann es bei der Synchronisation zu Änderungen am gleichen Datensatz in zwei oder mehreren Replikaten kommen.

Es wird unterschieden zwischen:

Eindeutigkeitskonflite (Uniqueness Conflicts) entstehen, wenn ein Datensatz an verschiedenen Replikaten erzeugt wurde und der nun an einem anderen Replikat eingefügt werden soll, in dem dieser Datensatz mit dem Primärschlüssel oder ein Attribut, das als eindeutig definiert ist, schon existiert.

Aktualisierungskonflikte (Update Conflicts) treten auf, wenn ein Datum von verschiedenen Replikaten geändert wurde und diese Änderungen zum selben Synchronisierungszeitpunkt vorliegen.

Löschkonflike (Delete Conflicts) entstehen, wenn der gleiche Datensatz auf dem einen Replikat bereits gelöscht wurde, auf einem anderen Replikat aber noch geändert oder als Fremdschlüssel verwendet wird. Diesen Konflikt kann man vermeiden, indem man die zu löschenden Datensätze im ersten Schritt nur als gelöscht kennzeichnet und dann versucht die Änderungen an allen Replikaten zu übernehmen und sie danach auf dem Replikat versucht zu löschen bzw. ihre Abhängigkeiten (referentielle Integrität) aufhebt.

Die Replikationskonflikte können durch einen automatischen Resolver[68] behandelt werden oder müssen manuell gelöst werden.

3.8 Replikationsverfahren

Im Folgenden werden einige gebräuchliche Replikations- und Synchronisationsverfahren beschrieben

3.8.1 Klassische, konsistenzerhaltende Verfahren[69]

Die klassischen synchronen Replikationsverfahren werden vor allem bei verteilten Datenbanksystemen eingesetzt. Wegen der engen Kopplung der beteiligten Systeme (hohe Netzwerkverfügbarkeit) und ihrer schlechten Skalier- und Erweiterbarkeit kommen sie in der Praxis bei der Replikation kaum mehr vor.

Wird die strenge Daten-Konsistenz für die Replikate gelockert, und gibt sich mit einer latenten Konvergenz zufrieden, sind losere Kopplungen der replizierten Systeme möglich. Erst

[68] Resolver wird hier als Konfliktauflöser verstanden und kommt bei Merge-Replikationen zum Einsatz
[69] Vgl. [Mutsch 04], p. 89 ff.

dadurch werden offline Lösungen wie mobile Datenbankanwendungen möglich, die sich nur selten mit den anderen Replikaten synchronisieren.

Zu den klassischen Verfahren der Synchronen Replikation gehören: [70]

- Read-One-Write-All-Verfahren (ROWA)
- Auswahl der Update-Kopie

Read-One-Write-All-Verfahren (ROWA):

Der einfachste Weg alle Replikate konsistent zu halten, ist es alle Replikate innerhalb derselben Transaktion zu aktualisieren. Dabei sind alle Replikate gleichzeitig gesperrt und somit können auch keine Inkonsistenzen beim Schreiben sowie beim Lesen auftreten. Jedes Replikat arbeitet und verfügt zu jedem Zeitpunkt über die aktuellen und konsistenten Daten.

Die Dauer der Transaktion für eine Änderung wird durch das am langsamsten arbeitende Replikat bestimmt. Jedes konventionelle Synchronisationsverfahren kann zum Zuge kommen um die Serialisierbarkeit (strikte Konsistenzbegriff: 1-Kopien-Serialisierbarkeit) zu garantieren, da alle Datenelemente auf den Replikaten normal erreichbar sind. Das ROWA Verfahren ist fehlerintolerant und für konsistente Leseoperationen ideal, da die Leselast gut verteilt werden kann. Für Schreibtransaktionen ist dieses Verfahren zu rigide, da alle Replikate hoch verfügbar sein müssen. In der Praxis bedeutet diese hohe Verfügbarkeit von geografisch getrennten Datenbanken grosse finanzielle Kosten.

Auswahl der Update-Kopie Verfahren:

Im Normalfall ist es nicht nötig alle Replikate als primäres Update Replikat auszuwählen, sondern es reicht aus, ein Replikat dafür zu bestimmen. Dieses Replikat wird dann Master Replikat genannt oder Primär Kopie. Alle anderen Replikate aus dem Replikatverbund sind dann sekundäre Kopien (Slaves) von der Primär Kopie. Eine Änderungstransaktion wird im ersten Schritt synchron an den Master weitergeleitet und im zweiten Schritt verteilt der Master dann die Änderungen an die Slave Replikate. Die Serialisierbarkeitsbedingung kann hier relativ einfach durch konventionelle Synchronisierungsverfahren eingehalten werden, da ja nur ein System von der Transaktion vom Beginn [71] bis zum Commit betroffen ist.

Das bekannteste Verfahren dazu ist das konsistenzerhaltende, pessimistische **Primary Copy (Single-Master):**

Es wird ein Replikat als Primär Kopie (Master) ausgewählt. Es ist ein asymmetrisches Verfahren mit synchronen Änderungen am Master Replikat. Alle Lesetransaktionen an den Replikaten greifen auf das lokale Replikat zu.

---

[70] Vgl.[Beut 96] p. 204

[71] Wird auch als BOT (begin of transaction) bezeichnet, oder 'begin tran'

Wie erwähnt erfolgt das Verfahren in zwei Stufen: Änderungs-Transaktionen, die von einem beliebigen Replikat (Slave) des Systems ausgelöst werden welches den Master sperren kann, ändern die Primärkopie. Der Master verwaltet die Schreib- und Lesesperren und propagiert die Änderungen asynchron an die Kopien. Bei der synchronen Variante können nach erfolgreicher Transaktionsdurchführung durch Update Trigger die Änderungen auf die Slaves sofort verteilt werden. Um eine Kaskadierung der Änderungen zu unterbinden müssen Ausnahmen zu den Triggern implementiert werden können. Der Nachteil dieses Verfahrens ist, dass bei Änderungstransaktionen eine Netzwerkverbindung zum Master hergestellt wird, und diese sind im unverbundenen Zustand nicht möglich. Weitere Nachteile sind der Ausfall des Master Replikats[72] und die Gewährleistung der Lesekonsistenz auf dem Slave, ob zum Zeitpunkt des lokalen Lesens schon alle relevanten Änderungen am Master durchgeführt wurden. Dazu kann eine Lesesperre vom Master Replikat angefordert werden, was aber dazuführt, dass der Vorteil einer erhöhten lokalen Lese Verfügbarkeit wieder sinkt.

3.8.2 Verfügbarkeitserhaltende asynchrone Verfahren[73]

In den gebräuchlichsten Replikationstopologien kommen asynchrone Replikationsverfahren zum Einsatz. Zwischen den Ursprungstransaktionsänderungen und der Replizierung auf die anderen Replikate liegt eine mehr oder weniger lange Latenz. Bei den asynchronen Verfahren wird Wert auf eine höhere Verfügbarkeit bei Leseoperationen zulasten von konsistenten Daten gelegt. Die Daten in den Replikaten sind nur zu dem Zeitpunkt der Replikation überall ident (synchron) und stellen somit nur eine Momentaufnahme des primären Replikats dar. Dadurch treten häufiger Konflikte auf und diese müssen oft durch semantisches Wissen oder Resolver gelöst werden.

Vertreter der asynchronen Replikation sind

- Virtual Primary Copy
- Schnappschussreplikation
- Merge-Replikation

Virtual Primary Copy

Eine Abwandlung des Primary Copy Verfahrens ist das Virtual Primary Copy, um auch in einem lokalen Replikat autonom Änderungen durchzuführen. Dazu wird von jeder realen Primär Kopie eine virtuelle Primärkopie auf einem zentralen Datenbanksystem erstellt. Ist die

[72] Wird auch als Single-Point-of-Failure bezeichnet, weil wenn der zentrale Punkt des Systems ausfällt, das ganze System ausfällt.
[73] Vgl. [Mutsch 04], p. 94 ff.

reale Primär Kopie für die anderen Replikate zwecks Änderungsupdate nicht erreichbar, so bedienen sich die anderen Replikate der virtuellen Primary Copy. Wenn die reale Primär Kopie wieder erreichbar ist, werden die Änderungen von der virtuellen Primary Copy zur realen Primär Kopie synchronisiert.

Snapshot oder Schnappschuss-Replikation

Snapshots[74] sind Sichten[75] auf eine Primärkopie und halten einen eingefrorenen Zustand fest. Bei der Generierung kann auch eine horizontale und vertikale Filterung der zugrunde liegenden Tabellen erfolgen. Das Snapshot-Verfahren wird oft nur für Updates von der primären Quelle auf dem Replikat verwendet. Die Aktualisierungshäufigkeit wird über den Snapshot Knoten bestimmt. Häufig dient der Snapshot auch zum erstmaligen Aktualisieren der Replikate, anstelle des Einspielens eines Datenbankdumps. Der Vorteil ist, dass die replikatsrelevanten Daten zuvor schon verdichtet werden können. Ein Snapshot enthält grundsätzlich das dem Replikat zugrunde liegende Schema und kann auch neben gefilterten Tabellen weitere Artikel wie im Datenbank Schema gespeicherte Prozeduren oder programmierte Funktionen, also Elemente aus der DML enthalten. Eine Snapshot-Replikation basiert meist auf dem Publisher-Subscriber-Modell (ähnlich dem Verlagswesen). Ein Verleger, Publisher generiert Schnappschüsse in einer Publikation. Über einen Distributor (Verteiler) werden die Publikationen zu den Abonnenten (Subscriber) verteilt. Der Snapshot kann synchron (verbindungsorientiert, sofort) oder asynchron (nach Aufforderung, oder zeitbasiert) zum Subscriber überspielt und angewendet werden. Eine Synchronisation kann als Abgleich von Snapshots verstanden werden. Beim Einspielen eines Snapshots wird die Zieltabelle immer gelöscht und neu erstellt. In Kapitel 5 wird die Snapshot Replikation im Detail vorgestellt.

Mergereplikation

Snapshot-Replikationen eignen sich nur für den lesenden Zugriff auf dem Replikat. Ist gefordert, dass von mehr als einer Seite Daten im unverbundenen Zustand autonom geändert werden sollen, kann die Merge-Replikation verwendet werden. Bei diesen dezentralen Updates werden Daten gemerged[76].

Beim Update-Everywhere Ansatz kann jedes Replikat auf jedem Knoten aktualisiert werden. Daher kann es zu Dateninkonsistenzen (Replikationskonflikt) beim Abgleich kommen und

[74] Im Zusammenhang mit der Replikation wird immer der Ausdruck Snapshots für Schnappschüsse gebraucht, da in der Lösung ebenfalls von Schnappschuss gesprochen wird, aber diese von der Lösung selber erstellt werden und nicht von den Datenbankreplikationsmechanismen

[75] im DB Kontext werden Sichten auch Views genannt

[76] D.h. Daten werden zusammengeführt, gemischt.

diese müssen durch geeignete Konfliktauflösungsverfahren behoben werden damit keine Änderungen verloren gehen (Lost Updates).

Dazu gibt es zwei Lösungsansätze:

- Partitionierung der Daten und
- Implementierung von Konfliktauflösungsverfahren

Konfliktvermeidung durch Partitionierung:

Ein Datensatz kann nur in jener Datenbank geändert werden, in der er auch erzeugt wurde. Dadurch bleibt auch die Transaktionskonsistenz erhalten. Viele Anwendungsfälle können durch diesen Ansatz recht gut gelöst werden. Z.B. ein Produktionsrezept kann nur am jeweiligen Standort Server, an dem es erstellt und wo es auch zum Produzieren verwendet wird, geändert werden.

Die Konfliktvermeidung kann in der Praxis allerdings kompliziert werden. Der MS SQL Server bietet dabei ein vereinfachtes Verfahren (Site-Priority) zur Konfliktlösung durch die Priorisierung einer Seite, des Publishers oder des Subscribers an. Wird nur der Publisher priorisiert heisst die Konfliktauflösung meist „Master/Slave - „Mastercopy wins“. Eine bessere Möglichkeit ist es, wenn der geänderte Datensatz mit einer Versionskennung versehen (Versionsnummer oder Timestamp) wird. Damit lassen sich im Replikat anhand der Versionsnummer des gleichen, schon enthaltenen Datensatzes erkennen, ob die Änderung übernommen werden soll oder nicht. Die Entscheidungskriterien können dazu ’latest timestamp’, ‚highest version’, ’earliest timestamp’ oder ’least Version’ sein.

3.9 Zusammenfassung

Generell gibt es zwei Arten von Replikationsverfahren, **synchrone** und **asynchrone**.
Synchrone Replikationsverfahren kommen vor allem bei verteilten Datenbanksystemen zum Einsatz. Sie sind in der Praxis aber kaum mehr von Bedeutung.
Asynchrone Replikationsverfahren werden in den heutigen typischen Anwendungsszenarien mit dezentralen, autonomen Datenbanken bevorzugt. Die losere Kopplung der replizierten Systeme wird erreicht, wenn man von der strikten Konsistenz abrückt und sie durch den schwächeren Konsistenzbegriff der **Konvergenz** ersetzt. Dies ermöglicht offline verfügbare Datenbanken und mobile Datenbanken ohne permanente Netzwerkverbindung zu den anderen Replikaten.

4 Anforderungen

4.1 Aktueller Stand der vorhandenen Software

Die TKT-Presta setzt seit einigen Jahren an ihren Nockenwellen Produktionsanlagen eine Server-Client basierte Rezepturverwaltung ein. Diese Rezepturverwaltung RZNW stammt vom Softwarehersteller ProgTec GmbH in Altstätten/Schweiz, bei der der Autor als Software-Ingenieur angestellt ist. Wie in der Einleitung erwähnt kann mit der für die TKT-Presta individuell entwickelten RZNW der Herstellungsprozess ihrer Produkte in Form von Rezepten gut beschrieben werden. Dazu können von den Prozess- und Qualitätsverantwortlichen die Produktionsablaufschritte mit ihren Bauteilen und ihren Parameter über das Server Frontend der RZNW auf der zentralen Standort Datenbank gepflegt werden. Abbildung 21 zeigt das Zusammenspiel der Systeme. An den Maschinenrechnern kann dann ebenfalls über die RZNW, die dann im Client-Betrieb läuft, die für die Anlage erstellten Rezepte auf die Maschinen Datenbank (MSDE[77]) synchronisiert werden. Der Maschinenbenutzer sendet vor Produktionsstart über die RZNW eine Art Parameterliste auf die Maschinensteuerungen. Die RZNW bietet für die Rezept-Konfigurationen unterschiedliche Editoren an. Die RZNW ist ursprünglich in Delphi 7 geschrieben und verwendet für die Datenbank-Verbindung zu einer ab Microsoft SQL Server 2000 Datenbank den 'Microsoft OLE DB Provider for SQL Server' Treiber.

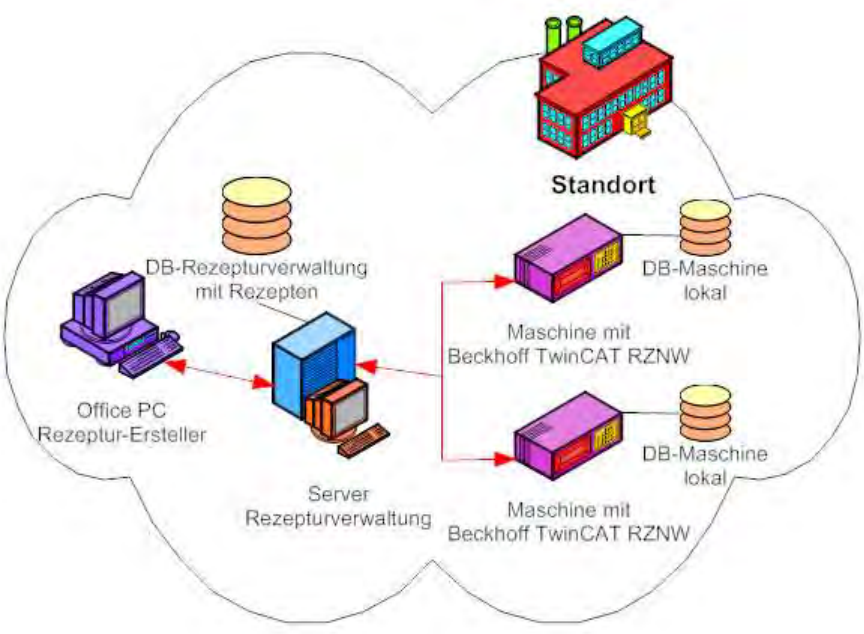


Abb. 21: RZNW-Struktur Landschaft an einem Standort

[77] MSDE = Microsoft SQL Server Data Engine, gratis Version des MS SQL Server 2000 für kleine Anwendungen

Hiervon wichtig für das Konzeptverständnis sind der 'Globale Rezeptur-Vorlagen'[78])- und der ‚Messdefinitionen/Rezeptur'[79] Editor.

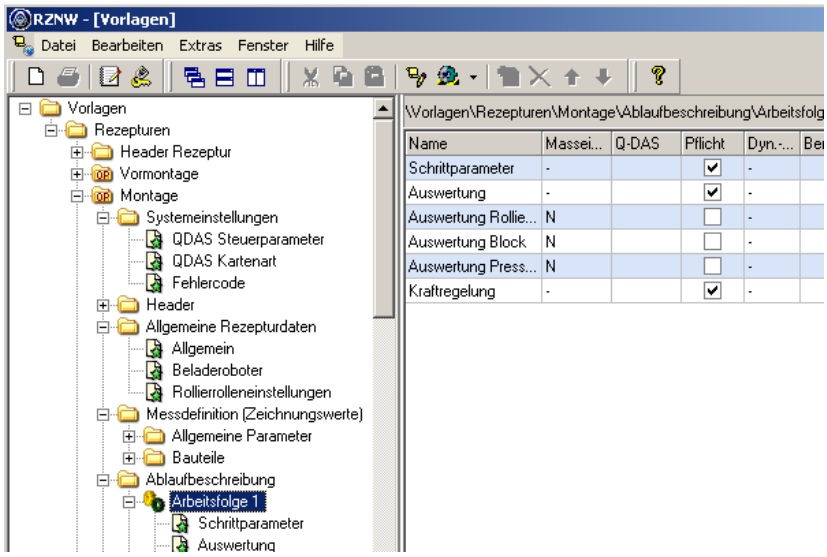


Abb. 22: RZNW-Vorlageneditor für die Konfiuration des Master (Maximal)-Rezepts

Über den Vorlagen-Editor werden die Strukturen, die Bauteile und die Parameter gepflegt, die dann anschliessend im Rezeptur Editor zu einem Rezept hinzugefügt und deren Parameterwerte dann unter Einhaltung der Wertegrenzen dort verändert werden können.

RZNW: Rezepturen folgen einem Strukturaufbau und bestehen daher aus verschiedenen Ordnern, mit den ihnen zugeordneten Merkmalen und diesen Merkmalen sind wiederum Parameter zugeordnet. Siehe Abbildung 23. Typischerweise sind in einem Rezept ca. 50-300 Ordner, mit ca. 1000-2000 Merkmalen und ca. 2000-5000 Parametern enthalten. An allen fünf Standorten zusammen sind ca. 3500 Rezepte (Stand 05. Mai 2011) angelegt.

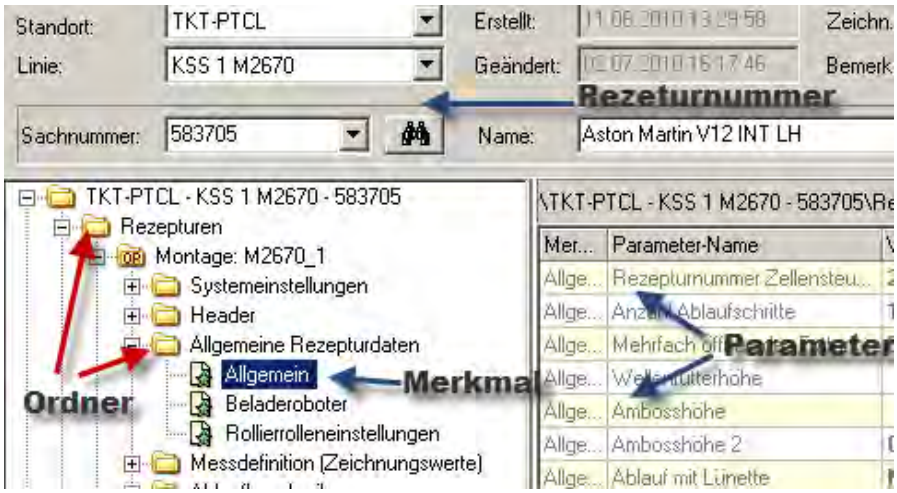


Abb. 23: RZNW-Rezeptur Strukturierungselemente

[78] 'Globale Rezeptur-Vorlagen' wird als 'Vorlagen' abgekürzt
[79] 'Messdefinitionen/Rezeptur' wird als 'Rezeptur' abgekürzt

Die Beziehungen zwischen z.B. den ’Parameter’ und den ‚Rezeptur’ Tabellen wird über Beziehungstabellen hergestellt, hier z.B. die ’bez_RezPrm’ und enthält ca. 2.5Mio Einträge (Stand 05. Mai 2011). Die RZNW Datenbank Struktur ist im Entity-Relationship Diagramm im Anhang 1 abgebildet.

4.2 Problemstellung und Kundenforderung

Die TKT-Presta produziert an mehreren Standorten gleiche bzw. fast identische Produkte. Dazu können derzeit Rezepte aus der RZNW an einem Standort[80] exportiert und an einem anderen STAO wieder importiert werden. Die Rezept-Vorlagen müssen dafür jedoch identisch sein, weil jedes Rezept auf Teilen der Ordner, Merkmale und Parameter aus den Rezept-Vorlagen basiert. Die Vorlagen können bis jetzt nicht einfach zwischen den STAOs ausgetauscht werden, bzw. nur über einen sehr zeitaufwändigen Ex-/ und Import. Deshalb wird eine Lösung gesucht, um die derzeit quasi autonom arbeitenden RZNW Server Datenbanken an den STAOs miteinander zu synchronisieren.
Die Hauptgründe für die neuen Anforderungen sind somit:

- Die Rezepte von einem STAO sollen auf allen anderen STAOs zur Verfügung stehen.
- An allen STAOs sollen nur mehr Rezepte erstellt werden, die auf unternehmensweit gleichen Rezept-Vorlagen basieren (Konsistenz).

4.3 Weitere Anforderungen

1) Es soll möglich sein, die Rezept-Vorlagen an allen STAOs zu erweitern und zu bearbeiten.
2) Sie sollen dann unter den STAOs abgeglichen werden können.
Anmerkung: Eine beliebige Erweiterung der Rezept-Vorlagen mit gegenseitiger (evtl. sogar automatischer) Übernahme aller Änderungen birgt das grosse Risiko eines unkontrollierten 'Wildwuchses' der Vorlagen (Inkonsistenz), sodass diese über die Zeit in einen nicht mehr zu handhabenden Zustand mutieren könnten.
3) Aus diesem Grund sollen die vorgenommenen Änderungen an den Vorlagen von einem Administrator an einem zentralen Hauptstandort geprüft und übernommen werden und er legt den Zeitpunkt der Verteilung an alle anderen Standorte fest.
4) Trotzdem muss es aber möglich sein, am lokalen Standort-Server neue Strukturen in die Vorlagen einzufügen und diese schon in Rezepte einzubinden. Ohne diese Möglichkeit wären die Reaktionszeiten auf kurzfristig notwendige Anpassungen viel zu lang.
5) Alle Änderungen an den Vorlagen und an den Rezepten werden derzeit mitprotokolliert. Es soll neu möglich sein, alle Protokolle zentral auf dem Hauptstandort zu sammeln.

[80] Standord wird mit STAO abgekürzt, Mehrzahl = STAOs

4.4 Derzeit bekannte Randbedingungen

1) Die Lösung soll so umgesetzt werden, dass möglichst wenig am RZNW Frontend geändert werden muss. Weiters sollen die RZNW Maschinen Datenbanken nicht von den Änderungen betroffen sein. Das bestehende Datenbankmodell der RZNW soll auch möglichst wenig geändert werden müssen.
2) Es soll keine zusätzliche Software von Drittherstellern eingesetzt werden müssen.
3) Das Übertragen der Daten soll durch die MS SQL Server >= 2000 datenbankeigenen Replikationsmechanismen erfolgen. Das Übertragen der modifizierten Daten z.B. in Änderungstabellen mittels Backup ist keine adäquate Lösung, da die Automatisierung der Synchronisierung im Vordergrund steht, bzw. eben möglichst vollautomatisch im Hintergrund ablaufen soll. Die Replikation ist nur ein Teilaspekt, ebenso wichtig sind die Konzeptionierung der Änderungsverfolgung und die kontrollierte Übernahme der Rezept-Vorlagenänderungen von den STAOs.
4) Die Synchronisierungen sollen zeitnah erfolgen, d.h. werden am Hauptserver die Vorlagen für die Verteilung freigegeben, dann sollen die STAOs innerhalb von maximal einem halben Arbeitstag synchronisiert sein.
5) Synchronisierungen werden vor allem im Zeitraum von Inbetriebnahmephase von neuen Produkten oder Maschinen erfolgen müssen, da dann Änderungen so schnell wie möglich an den Standorten zur Verfügung stehen sollen.
5) Rezepte werden bisher nur sporadisch ausgetauscht. Bei Maschinenverlagerungen zu anderen Standorten sollen auch alle Rezepte am neuen Standort zur Verfügung stehen.

4.5 Beschreibung der vorhandenen Systeme

Alle Standorte sind über ein VPN (virtual private network) miteinander verbunden, wobei es je nach STAO und Tageszeit zu starken Netzwerk-Bandbreiten-Schwankungen kommt. Der Standort in Dalian - China verfügt über die langsamste Netzwerk-Verbindung mit einer gemessenen mittleren Geschwindigkeit von 50kBit/s (ca. 540Mb/24h).

Bei drei Maschinenlieferanten stehen Rechner mit Windows 2000 Server Betriebssystem und SQL Server 2000 SP4 Standard Edition. Diese Rechner sind meistens offline und über das Netzwerk nur nach Absprache erreichbar.

An vier Produktionsstandorten kommt Windows Server 2003 R2 mit SQL Server 2005 Standard Edition zum Einsatz.

Der Hauptserver in Eschen wird mit Windows Server 2008 und dem SQL Server 2008 Standard Edition betrieben.

5 Konzept

Um das Konzept besser zu verstehen, ist es wichtig, das RZNW ERD aus Anhang 1 zu kennen. In ihm sind die Beziehungen zwischen den Tabellen dargestellt. Ausgehend von diesem ERD wurde die Lösung entwickelt und die Aufteilung der Tabellen in die einzelnen Replikationspublikationen vorgenommen.

In Abbildung 21 ist die Systemlandschaft eines einzelnen RZNW Standorts (in weiterer Folge immer STAO genannt) abgebildet, die bei allen Standorten identisch ist. Als zentraler Synchronisierungspartner ist neu ein Hauptstandort zu definieren. Die Projektverantwortlichen und Administratoren für die RZNW arbeiten in Eschen – Liechtenstein und deshalb wird dort der zentrale RZNW Hauptstandort Datenbank Server (in weiterer Folge immer HS oder Hauptserver genannt) der Synchronisierungslösung eingerichtet. Die neue Lösung soll RZNW-Multiserver heissen.

Für den Datentransport zwischen den autonomen RZNW Standort Datenbanksystemen kommt nur die **asynchrone Replikation** in Frage, da N Datenbanken über eine Master Datenbank zu unterschiedlichen, benutzerdefinierten Zeitpunkten synchronisiert werden sollen. Alle STAOs sollen ihre autonomen Änderungen am HS einfügen. Deshalb muss geklärt werden, ob und in welcher Art der MS SQL Server asynchrone Replikationsverfahren unterstützt. ACHTUNG: da hier unterschiedliche SQL Server Versionen im Einsatz sind, muss das Konzept so ausgelegt werden, dass die Lösung möglichst alle Systeme ohne grosse Adaptierungen von einem System zum Anderen unterstützt. Es gibt gerade in Bezug auf automatische Datenänderungsverfolgung (change data capture (Historie), change tracking) am SQL Server 2008 gute Lösungsansätze (die Tests haben aber gezeigt, dass auch diese Funktionen in Bezug auf die Anforderungen beim RZNW-Multiserver nur Grundfunktionen abdecken und kaum nützlich waren).

5.1 Replikationsverfahren im MS SQL Server

Der MS SQL Server ab Version 2000 bietet folgende Replikationstypen an:

1) Transaktions-Replikation.
2) Merge-Replikation.
3) Snapshot-Replikation.

Microsoft schreibt, dass die Entscheidung für einen Replikationstyp von vielen Faktoren abhängt.

- Von der physikalischen Replikationsumgebung:

Es sind die Anzahl und der Standort der an der Replikation beteiligten Computer zur berücksichtigen (Server, Client Rechner, Laptops oder mobile Geräte (Smartphones)

- Die Art und die Menge der zu replizierenden Daten:

Und ob auch die Daten auf dem Abonnent (Replikat) aktualisiert werden (also, ob Daten autonom am Replikat geändert werden können ohne diese Update Transaktionen synchron am Master auszuführen).[81]

5.2 Begriffe für die Replikation mit dem MS SQL Server

[MSRep 11] definiert

„Bei der Replikation wird zur Darstellung der Komponenten in einer Replikationstopologie - Verleger, Verteiler, Abonnenten, Veröffentlichungen, Artikel und Abonnements - ein Modell verwendet, das sich auf Abläufe aus dem Verlagswesen stützt. Die Microsoft SQL Server-Replikation funktioniert dabei so ähnlich wie ein Zeitschriftenabonnement:

- Ein Zeitschriftenverlag (Verleger) stellt eine oder mehrere Zeitschrift(en) (Veröffentlichungen) her.
- Jede Veröffentlichung enthält verschiedene Artikel.
- Der Verlag verteilt die Veröffentlichung direkt oder über eine Vertriebsorganisation (Verteiler).
- Abonnenten erhalten genau die Veröffentlichungen, die sie abonniert haben.

[…]“[82]

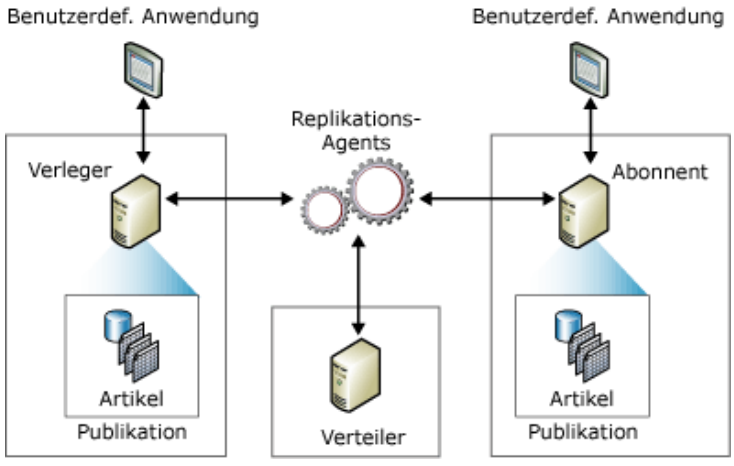


Abb. 24: Komponenten und Prozesse die bei der Replikation beteiligt sind[83]

[…]

[81] Vgl. /ReTy 11/
[82] /MSRep 11/
[83] /MSRep 11/

„Die Replikationstopologie definiert die Beziehung zwischen Servern und die Kopien von Daten sowie die Logik, die den Datenfluss zwischen den Servern festlegt. Für das Kopieren und Verschieben von Daten zwischen dem Verleger und den Abonnenten sind eine Reihe von Replikationsprozessen (so genannte Agents) verantwortlich. Die folgende Abbildung gibt eine Übersicht über die Komponenten und Prozesse, die bei der Replikation beteiligt sind.“[84]

5.2.1 Verleger

Der Verleger (Publisher) ist eine Datenbankinstanz, der anderen Datenbank Replikaten eine oder mehrere Publikationen zur Verfügung stellt. In einer **Publikation** werden Artikel wie Tabellen, Sichten (Views), DML Elemente wie Funktionen oder Stored Proecdures aus einer Datenbank als ein logisches Objekt zusammengefasst. [85]

5.2.2 Verteiler

Der Verteiler (Distributor) kann aus Performanzgründen in einer eigenen Datenbankinstanz liegen (Remote Distributor). Jedem Publisher ist ein Distributor zugeordnet und er dient dazu deren Publikationen in seiner Datenbank als Replikationsdaten und Metadaten für die Abonnenten (Subscriber) vorzuhalten und an sie zu verteilen. Das kann über direkten Dateizugriff oder über FTP erfolgen. Deshalb ist auch der MS Internet Information Services (IIS) notwendig, der einen FTP Server zur Verfügung stellt. [86]

5.2.3 Abonnenten

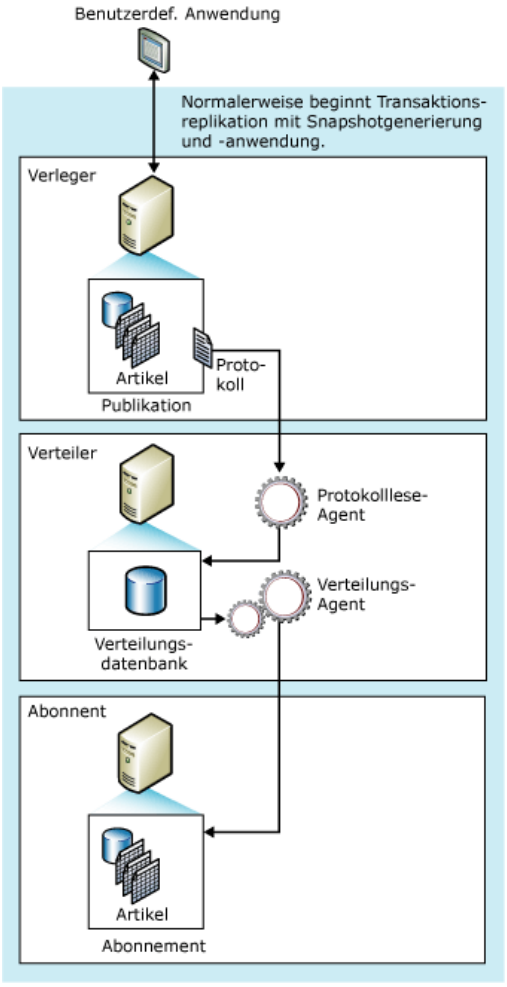
Ein Abonnent (Subscriber) ist eine Datenbankinstanz, die replizierte Daten als Abonnement (Subscription) empfängt. Abonnenten können Subscriptions auf mehrere Publisher und Publikationen haben. Je nach ausgewähltem Replikationstyp kann der Abonnent auch Datenänderungen an den Verleger zurücksenden oder die Daten erneut für andere Abonnenten veröffentlichen, dann wird er selber zum Publisher. Bei einer **Push Subscription** erstellt der Verleger von sich aus die Publikationen, wohingegen über eine **Pull Subscription** der Abonnent beim Verleger die Erstellung einer neuen Publikation anfordern kann (vor allem bei Merge-Replikationen).[87]

[84] Vgl. ibid
[85] Vgl. ibid
[86] Vgl. ibid
[87] Vgl. ibid

5.2.4 Transaktionsreplikation[88]

Die Transaktionsreplikation wird typischerweise in reinen Serverumgebungen verwendet und ist für die folgenden Fälle geeignet:

- Inkrementelle Änderungen sollen quasi synchron an Abonnenten weitergegeben werden.
- Änderungen am Verleger stehen mit geringer Latenzzeit beim Abonnenten zur Verfügung -> hoher Durchsatz.
- Die Anwendung möchte auch die zeitliche Abfolge der Datenänderungen an einem Datensatz mitbekommen. Wenn eine Zeile sich z. B. fünfmal ändert, kann die Anwendung bei Verwendung der Transaktionsreplikation auf jede Änderung reagieren (indem sie z. B. einen Trigger auslöst), nicht nur auf das endgültige Ergebnis aller Zeilenänderungen.
- Beim Verleger werden häufig Datenänderungen vorgenommen.



Ablauf

Der Protokolllese Agent liest das Transaktionsprotokoll der Verleger Datenbank mit und kopiert nur die relevanten, auf die markierten Daten angewendeten, abgeschlossenen Transaktionen zum Distributor. Je nach Ausführungszeitpunkt führt der Distribution-Agent die Transaktionen fortlaufend (minimale Latenzzeit) oder in geplanten Intervallen auf den Abonnenten aus. Bei der verzögerten Aktualisierung über eine Warteschlange können Konflikte an der Abonnenten Datenbank auftreten, da sie sich zwischenzeitlich geändert hat.

Abb. 25: Ablauf der Transaktionsreplikation[89]

[88] Vgl. /MSRep 11/
[89] /ReTa 11/

5.2.5 Mergereplikation[90]

Die Mergereplikation wird typischerweise in Server/Client-Umgebungen verwendet. Ihre Verwendung empfiehlt sich in den folgenden Situationen:

- Mehrere Abonnenten aktualisieren dieselben Daten zu verschiedenen Zeitpunkten und geben diese Änderungen an den Verleger und an andere Abonnenten weiter. (bidirektionaler Abgleich)
- Abonnenten müssen Daten empfangen, Änderungen offline vornehmen und Änderungen später mit dem Verleger und anderen Abonnenten synchronisieren.
- Jeder Abonnent benötigt eine andere Datenpartition.
- Es kann zu Konflikten kommen, und Sie müssen in der Lage sein, die Konflikte zu erkennen und zu lösen.
- Die Anwendung muss nicht auf die einzelnen Zwischenstufen von Datenänderungen, sondern nur auf das endgültige Ergebnis der Datenänderung zugreifen können.

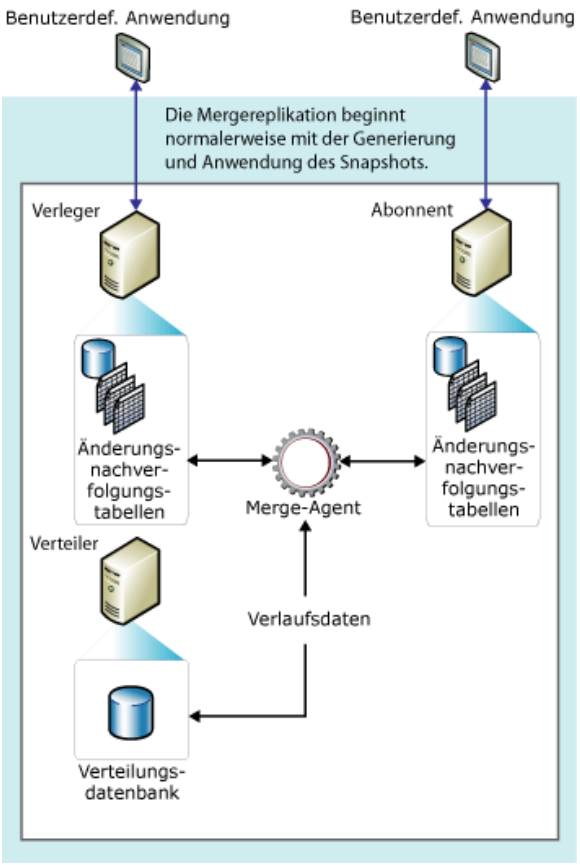


Abb. 26: Ablauf der Merge-Replikation[92]

Funktionsweise

Bei der Mergereplikation erfolgt die Änderungsnachverfolgung mithilfe von Triggern und Metadatentabellen. Die Mergereplikation ermöglicht es verschiedenen Standorten, autonom zu arbeiten und die Aktualisierungen zu einem späteren Zeitpunkt zu einem einzelnen, einheitlichen Ergebnis zusammenzuführen. Da Aktualisierungen auf mehr als einem Knoten ausgeführt werden, sind dieselben Daten möglicherweise vom Verleger und von mindestens einem Abonnenten aktualisiert worden. Daher kann es beim Zusammenführen von Aktualisierungen zu Konflikten kommen, für deren Lösung die Mergereplikation aber eine Reihe von Möglichkeiten bietet.[91]

[90] Vgl./ReMr 11/

[91] ibid

[92] ibid

5.2.6 Snapshotreplikation[93]

Datenänderungen auf dem Abonnenten werden bei der Snapshotreplikation nicht nachverfolgt. Alle beim Abonnent vorhandenen Artikel aus der Publikation werden komplett mit dem neuen Snapshot überschrieben.

Die Verwendung der Snapshotreplikation als eigenständiger Replikationstyp empfiehlt sich vor allem dann, wenn eine oder mehrere der folgenden Bedingungen erfüllt sind:

- Es kommt nur selten zu Datenänderungen, wenn dann sind sie umfangreich
- Es ist akzeptabel, dass Kopien der Daten bezüglich des Verlegers für eine bestimmte Zeit nicht aktuell sind.
- Es werden kleine Datenmengen repliziert.
- Große Mengen von Daten werden innerhalb einer kurzen Zeit geändert.

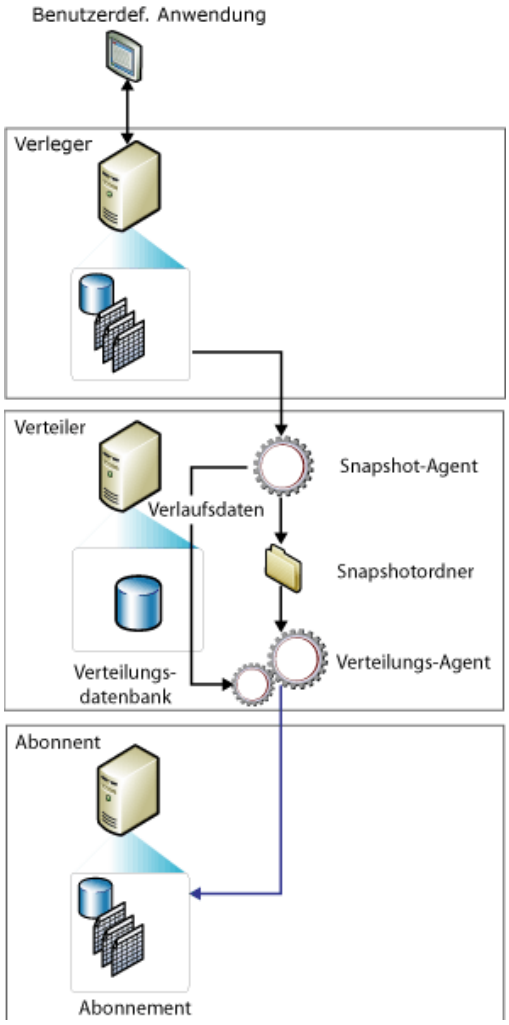


Abb. 27: Ablauf der Snapshot-Replikation[95]

Funktionsweise

Bei der Snapshot Erstellung wird der Zustand der Artikel (Tabellen usw.) am Verleger zum Erstellungszeitpunkt eingefroren, es wird ein Schnappsschuss gemacht. Wenn die Tabellen sehr gross sind, kann es zu längeren Sperren kommen. Der Abonnent ist also immer nur so aktuell, wie der Verleger zum Zeitpunkt der Schnappschusserstellung war.[94]

[93] Vgl./ReSr 11/
[94] ibid
[95] ibid

5.3 Kriterien zur Entwurfsentscheidung

5.3.1 Synchronisierungsrichtungen

Das RZNW System besteht aus einem zentralen HS und mehreren sternförmig vernetzten STAOs. Die STAOs kommunizieren nur über den HS und nicht unmittelbar miteinander, deshalb können zwei Synchronisierungsrichtungen unterschieden werden:

HS → STAO und ***STAO → HS***

5.3.2 Synchronisierungsmethoden/-funktionen

Können in beide Richtungen die gleichen Synchronisierungsfunktionen eingesetzt werden?

Für die Rezepte ja, da alle STAO Rezepte zum HS kommen und von dort alle Rezepte auf die anderen STAOs verteilt werden (derzeit keine Standortpartitionierung).

Für die Protokolllogbücher (nur mehr Logbücher genannt) nein. Es sollen alle Logbücher am HS zusammenkommen, aber nicht weiter auf die STAOs verteilt werden.

Für die Vorlagen nein. Es sollen nur Vorlagenänderungen vom STAO zum HS kommen, währenddessen sollen am STAO die Änderungen schon in Rezepten verwendet werden können. Nachdem am HS die Vorlagenänderungen von einem Administrator übernommen oder zurückgewiesen wurden, sollen nach der Freigabe der geänderten Vorlagen diese auf die STAOs synchronisiert werden.

5.3.3 Aufteilung der Synchronisierung:

Aus diesen Forderungen lassen sich mehrere Synchronisierungsaufgaben ableiten. D.h. die Daten der RZNW müssen vertikal segmentiert werden. Allerdings gilt es zu beachten, dass die Tabellen über Relationen miteinander in Beziehung stehen und die Konsistenz wird über die referentielle Integrität gewährleistet. Auch aufgrund der begrenzten Netzwerkbandbreite zu einigen STAOs ist eine Datenlastreduktion durch Aufspaltung der gesamten Änderungsdaten sinnvoll. In weiterer Folge muss eine mögliche Aufteilung gefunden werden.

Eine zentrale Anforderung ist, dass Vorlagenänderungen von den STAOs am HS zuerst auf ihre Korrektheit untersucht werden sollen und dann erst in die zentrale Datenbank am HS übernommen werden sollen. Dazu ist eine GUI Applikation notwendig, in der ein Bediener entscheidet, welche Änderungen übernommen und welche abgelehnt werden sollen. Dieses Gui kann auch für die manuelle Ausführung der Synchronisierung oder Logbücher, der Rezepte und der Vorlagen verwendet werden. Das GUI ist nicht Bestandteil dieser Arbeit.

5.4 Konzeptentwurf RZNW-MultiServer

Das Gesamtsystem 'RZNW' ist neu strukturiert in einen zentralen HS und mehrere verteilte STAOs. Um einerseits die Sperrzeiten der einzelnen Tabellen an der produktiven Datenbank kurz zu halten und um eine manuelle Übernahme durch den Anwender zu ermöglichen werden neue Datenbanken hinzugefügt. Einerseits dienen Export Datenbanken dazu um die Änderungsdaten von den zentralen Datenbanken einzufügen und von denen die Snapshots erstellt werden. Andererseits dienen die Import Datenbanken dazu, dass die Änderungen erst nach Bedarf übernommen werden können.

Die Server tauschen untereinander Informationen per DB-Replikation aus. Dabei werden - je nach Anwendungsfall – die Daten per Snapshot oder Mergereplikation ausgetauscht. Die folgende Abbildung stellt die Kommunikationsverbindungen eines einzelnen STAOs mit dem HS symbolisch dar:

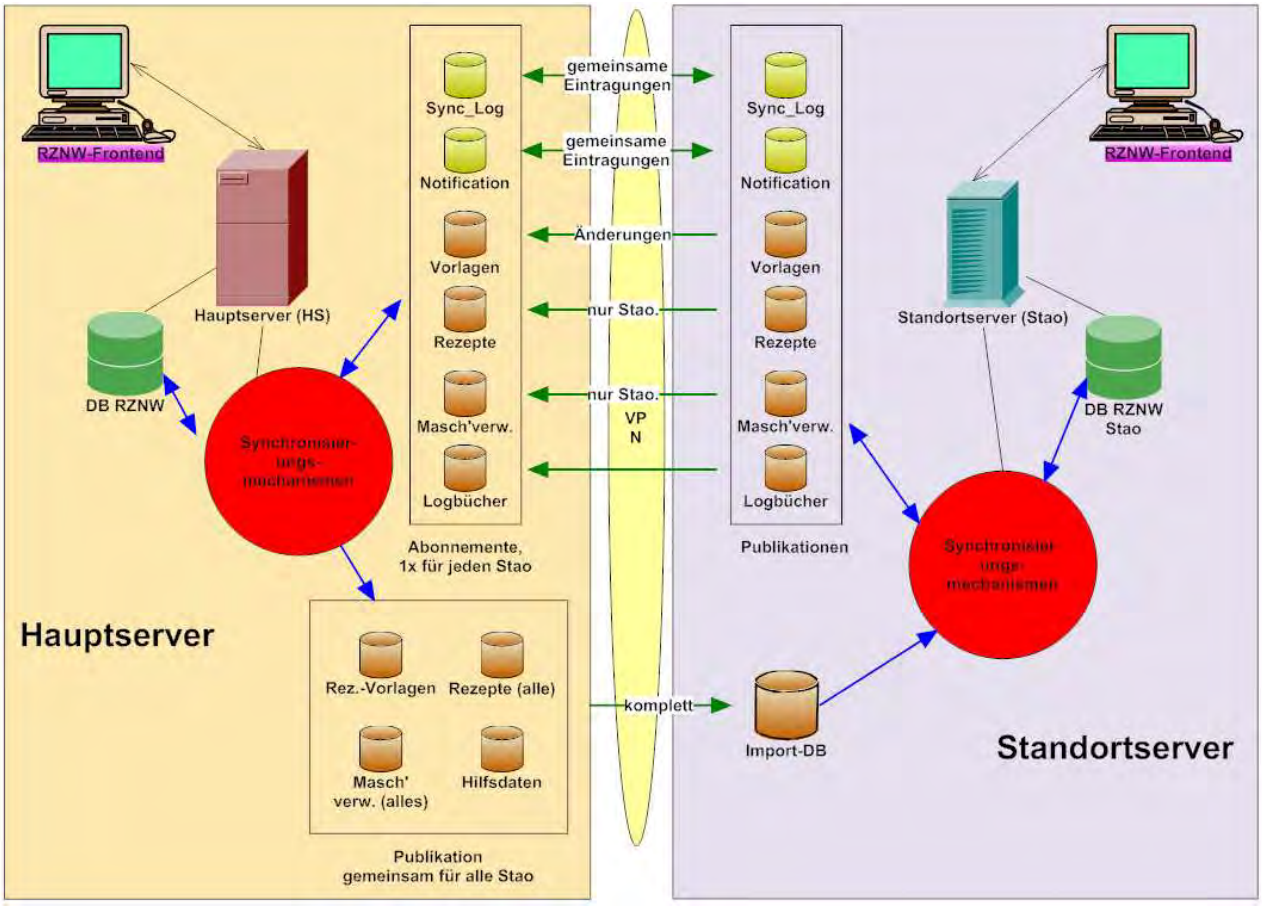


Abb. 28: Konzept RZNW Multiserver

5.4.1 Kommunikation HS → STAO

Der HS publiziert ein (fast) komplettes Abbild seiner DB über eine Snapshot-Replikation auf die beteiligten STAOs. Nicht publiziert werden die Logbücher. Wie in der Abbildung 28 symbolisch dargestellt wird die DB auf Grundlage des RZNW ER-Diagramms unterteilt in verschiedenen Publikationen zur Verfügung gestellt:

- Rezeptur-Vorlagen
- Rezepte
- Maschinenverwaltung

Diese Unterteilung hat den Hintergrund, dass die einzelnen Bestandteile der RZNW DB getrennt voneinander repliziert werden können, um Netzwerk Bandbreite zu sparen und die verschiedenen Publikationen unterschiedlich priorisieren zu können. Export- und Import-Datenbanken werden nun deshalb benötigt, weil die Synchronisierung auch manuell durchgeführt werden kann. Bei der Replikation mit dem MS SQL Server erfolgt der Austausch der Daten über Dateien. Entweder per Zugriff auf einen Netzwerkshare oder über FTP. Tests haben ergeben, dass für einen Vorlagenschnappschuss ca. 60MB und für einen Rezeptschnappschuss ca. 250MB grosse Datendateien erstellt werden.

Der publizierte DB-Schnappschuss des HS wird durch einen Administrator manuell ausgelöst und bleibt bestehen, bis er durch einen neuen Schnappschuss überschrieben wird. Die manuelle Auslösung garantiert einen konsistenten Zustand der replizierten Daten.

Die STAOs holen sich per Pull-Abonnement automatisch den aktuellen Snapshot ab und replizieren diesen auf ihre Import-DB. Von dort schliesslich können die aktualisierten Daten durch den lokalen Administrator manuell in die Standort-DB übernommen werden. Die manuelle Auslösung der Datenübernahme garantiert auch hier die Kontrolle über die Daten auf dem STAO Server.

Innerhalb jeder Publikation des HS wird auch der Zeitstempel des Schnappschusses zur Verfügung gestellt. Auf den STAOs wird dieser beim Import der Daten registriert.

Alle Import- und Export-Aktionen werden in einem Synchronisations-Log gespeichert und können so bei Problemen nachvollzogen werden.

5.4.2 Kommunikation STAO → HS

Jeder STAO publiziert Teile seiner Datenbank zur Replikation auf den HS. Folgende Publikationen stehen zur Verfügung:

- Vorlagen (nur Änderungen und Erweiterungen)
- Rezepte (nur lokale Rezepte)

- Maschinenverwaltung (nur lokale Daten)
- Logbücher (nur lokale Logbücher)

Die oben genannten Publikationen umfassen jeweils nur Daten in Richtung HS und werden per Snapshot-Replikation auf den HS übertragen (sie werden nachfolgend auch kurz STAO-Publikation genannt).

Ausserdem definiert der STAO zusätzliche Publikationen, welche jeweils als bidirektionaler Kommunikationskanal zwischen HS und STAO dienen. Da hier beide beteiligten Seiten an den Daten Änderungen vornehmen können, kommt für diese Publikationen eine Merge-Replikation zur Anwendung.

'Sync_Log'	In die Sync_Log Merge-Publikation werden von beiden Seiten die Ereignisse betreffend der Datensynchronisation eingetragen - also beispielsweise wenn auf dem STAO manuell ein Schnappschuss ausgelöst wird, um Erweiterungen an den HS zu übertragen.
'Notification'	Über diese Publikation informiert der HS den STAO über den aktuellen Status der lokalen Änderungen – beispielsweise welche Vorlagen-Erweiterungen angenommen bzw. abgelehnt worden sind (dazu weiter unten mehr).

Tabelle 5: Merge Publikationen

Auf dem HS existiert für jeden STAO eine eigene Import-DB, in welche per Abonnement die Publikationen des jeweiligen STAOs repliziert werden. Um den HS von den Kommunikationsaufgaben weitgehend zu entlasten, werden die Abonnemente als Push-Abonnemente auf den STAO realisiert und folglich von diesen abgewickelt. Publiziert wird generell auf dem STAO, auch bei den Merge-Replikationen.

Der Umstand, dass die Kommunikation hin zum HS über Publikationen des STAOs abgewickelt wird, bedingt, dass ein neuer STAO bei dessen Inbetriebnahme zwingend am HS eingerichtet werden muss. Dabei wird auf dem HS die Import-DB des neuen STAOs angelegt, auf welche dieser seine Daten repliziert (pushed). Die Import DB und Export DB weisen das gleiche Schema wie die RZNW Datenbanken auf mit dem Unterschied, dass keine referentielle Integritäten definiert wird (Constraints entfallen) und Trigger auf Tabellenänderungen definiert sind.

Der detaillierte Ablauf der Kommunikation per Log sowie die Übernahme der Standort-Daten am HS wird in den nachfolgenden Abschnitten beschrieben.

5.5 Funktionsweise

5.5.1 Rezept-Vorlagen

Die aktuell gültigen (genehmigten) Rezept-Vorlagen werden auf dem HS verwaltet. Die STAO können einen aktuellen Schnappschuss davon autonom vom HS beziehen. Der Bezug erfolgt automatisch per Pull-Abonnement.

An jedem STAO können die aktuellen Vorlagen erweitert werden. Die Erweiterungen gelten aber als provisorisch und können nur an diesem Server verwendet werden. Eine direkte Weitergabe (Export/Import) von Rezepten, welche die Erweiterungen einsetzen, ist in diesem Stadium nicht möglich.

Die lokal vorgenommenen Erweiterungen/Änderungen können jederzeit an den HS weitergeleitet werden (manuell ausgelöst). Der STAO aktualisiert dabei den Schnappschuss seiner Vorlagen-Publikation und trägt die Liste aller neuen bzw. geänderten Elemente in die 'Notification'-Publikation ein. Diese stellt also immer eine Liste aller neuen bzw. modifizierten Elemente dar. Diese Liste allein genügt am HS allerdings nicht zur Bearbeitung der Änderungen durch den Administrator; dazu ist die komplette Struktur nötig, welche im separaten Vorlagen-Schnappschuss geliefert wird. Die 'Notification'-Publikation liefert lediglich eine zusammengefasste Liste der Änderungen sowie den Kommunikationskanal für die Rückmeldung des HSs an den STAO über die übernommenen oder abgewiesenen Vorlagenänderungen.

Auf dem HS werden die an den STAOs durchgeführten Änderungen/Erweiterungen in der RZNW-Multiserver GUI aufgelistet und können zur besseren Erkennbarkeit zusätzlich auch in der gleichen Baumansicht wie im Vorlagen-Editor dargestellt werden. Sie werden von einem Administrator geprüft und entweder akzeptiert oder zurückgewiesen.

Bei Änderungen (nicht Erweiterungen) bestehender Elemente (z.B. anderer Name, Datentyp usw.) kann es prinzipiell zu Konflikten kommen, wenn das Element gleichzeitig an mehreren Standortservern geändert wurde. Da am HS vom Administrator die entsprechenden Änderungen sowieso nacheinander bearbeitet werden, entscheidet er über jede Änderung einzeln.

Am HS akzeptierte Änderungen/Erweiterungen stehen fortan allen Standortservern zur Verfügung (→ definitive Änderung/Erweiterung).

Alle direkt in der RZNW am HS vorgenommenen Änderungen/Erweiterungen gelten wie bisher sofort als akzeptiert (definitiv).

Das Ergebnis der Prüfung/Bearbeitung der Vorlagen am HS wird dem Quellen-STAO der Änderungen über seine 'Notification'-Publikation mitgeteilt. Dieser übernimmt die behandelten Elemente dann als akzeptiert oder markiert sie als zurückgewiesen. Das Update der 'No-

tification'-Publikation erfolgt unmittelbar nach dem Akzeptieren bzw. Zurückweisen durch den Administrator.

Der STAO prüft die 'Notification'-Publikation auf erledigte Einträge und übernimmt den neuen Status in seine Vorlagen. Nach der Übernahme löscht der STAO den entsprechenden Eintrag aus der 'Notification'-Publikation.

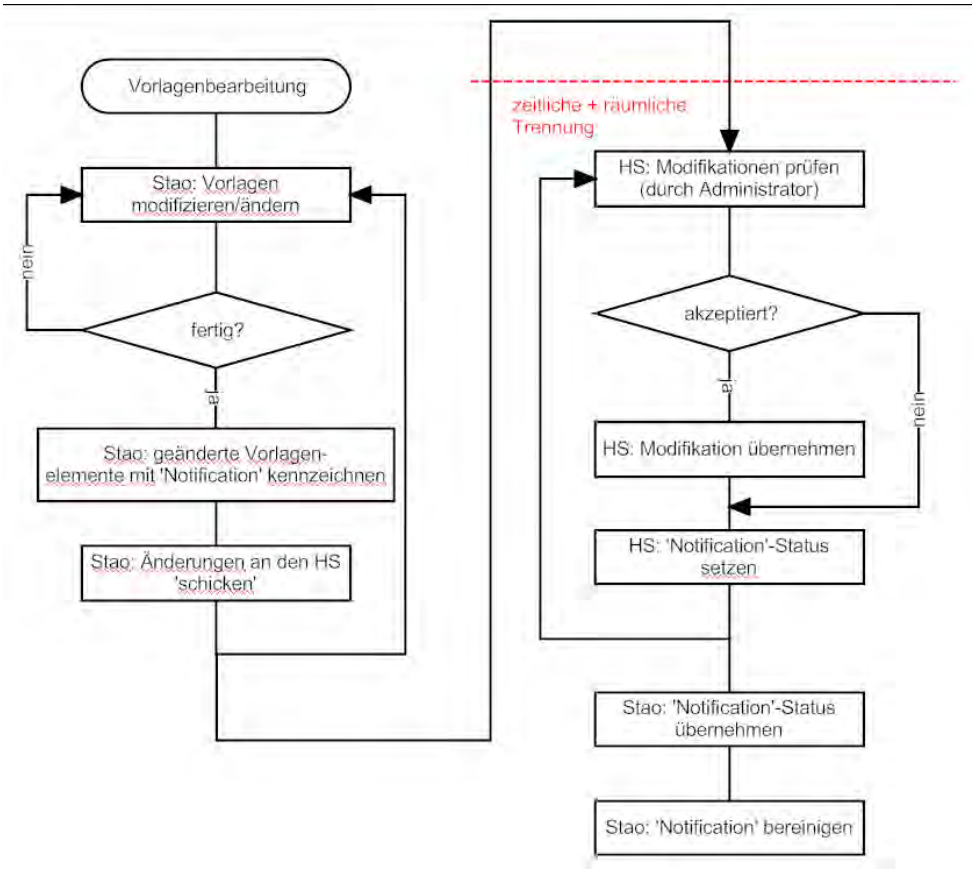


Abb. 29: funktioneller Ablauf der Vorlagensynchronisierung

Die zurückgewiesenen provisorischen Änderungen/Erweiterungen bleiben provisorisch und müssen in den Rezepten ggfs. durch andere, definitive Strukturen ersetzt und danach gelöscht werden. Damit sie nicht erneut am HS aufscheinen, werden sie in der Standort-DB entsprechend markiert.

Wurde ein bestehendes Element, welches auf einem STAO den Status 'provisorisch' trägt auch an einem anderen STAO bearbeitet und am HS akzeptiert, so liegt ein Konflikt vor.

Aufgrund eines Zeitstempels erkennt der STAO, ob die Änderung vom HS neuer ist als seine eigene. Falls ja, übernimmt er diese. Falls nein, behält er seine Änderung und den Status 'provisorisch'. Dadurch bleibt die Änderung für die nächste Übernahme-Aktion durch den Administrator am HS erhalten.

Erweiterungen und Änderungen an den Vorlagen, welche von anderen STAOs oder vom HS selbst stammen, erhält ein STAO über die allgemeine Vorlagen-Publikation des HS und diese werden komplett übernommen.

Am Standort geänderte, provisorische Elemente werden im RZNW-Frontend blau markiert, abgelehnte Elemente rot, so dass ihr Status ersichtlich ist.

5.5.2 Allgemeine Maschinenparameter

Es gibt in der Rezeptur Parameter, die nicht vom zu produzierenden Produkt abhängig sind, sondern nur von der Maschine auf der sie produziert werden. Die Struktur der nicht produktabhängigen (allgemeinen) Maschinenparameter zählt thematisch zu den Vorlagen und wird mit den Vorlagen mitgeliefert.

5.5.3 Maschinenverwaltung

Zusätzlich zu den Rezepturvorlagen und den Rezepturen gibt es eine Möglichkeit die Produktionsanlagen zu verwalten. Dies ist auch deshalb notwendig, weil Rezepte auch ganz genau einer Produktionsanlage zugeordnet werden müssen, auf der sie produziert werden. In der Maschinenverwaltung wird z.B. die Art der Kommunikation zwischen RZNW und SPS ausgewählt (z.b. Beckhoff TwinCat, Siemens Simatic SPS usw.)

Am STAO können an der Maschinenverwaltung nur innerhalb des eigenen Standorts Änderungen vorgenommen werden.

Änderungen auf dem STAO werden per Stao-Publikation an den HS übertragen. Der Ablauf ist identisch wie bei den Rezept-Vorlagen und können am HS automatisch oder manuell übernommen werden.

5.5.4 Rezepte

Auf dem STAO können nur Rezepte des eigenen Standorts erstellt und bearbeitet werden. Rezepte anderer Standorte sind 'readonly'. Sie können aber als Kopier-Vorlage für neue, eigene Rezepte verwendet werden.

Rezepte von STAO können am HS nicht bearbeitet werden und sind ‚readonly'.

Die STAO liefern ihre Rezepte per Stao-Publikation an den HS.

Der HS 'sammelt' die Rezepte aller STAOs und verteilt sie per HS-Publikation an alle anderen STAOs.

Neue und geänderte Rezepte von den Standorten können am HS automatisch oder manuell übernommen werden.

Die Änderungen an Rezepten anderer Standorte (aus HS-Publikation) können am STAO automatisch oder manuell übernommen werden.

5.5.5 Logbücher

Die STAOs liefern ihre Logbücher per Stao-Publikation an den HS.

Der HS 'sammelt' die Logs aller Standorte und fasst sie zentral zusammen.

Die Logs werden vom HS nicht publiziert → an den STAOs können nur die eigenen lokalen Logs eingesehen werden.

Damit die Stao-Publikation für die Logbücher nicht unkontrolliert wächst, werden nur die neuen Einträge publiziert, welche vom HS noch nicht übernommen worden sind. Der HS teilt dem STAO per Eintrag in die 'Sync_Log'-Publikation mit, bis zu welchem Zeitstempel die Log-Einträge übernommen wurden.

Die Logbücher von den STAOs können am HS automatisch oder manuell übernommen werden.

6 Realisierung

6.1 Notwendige Datenbank-Erweiterungen

Tabelle	Beschreibung
LocalModifiedElements	Elemente aus den Vorlagen, welche lokal geändert oder gelöscht wurden, erhalten in dieser Tabelle genau einen Eintrag. Aus Gründen der Trigger-Kompatibilität (gleicher Code) wird diese Tabelle auch auf dem Hauptserver geführt, obwohl am HS kein Delta-Schnappschuss über die geänderten Vorlagen erstellt wird.
LocalRemovedElements	Gelöschte Elemente aus allen Tabellen, welche in individuellen HS-Schnappschüssen an die Standorte repliziert werden. Aus Gründen der Trigger-Kompatibilität (gleicher Code) heisst diese Tabelle am HS gleich wie an den Stao. Der HS überträgt Timestamp-basiert die Infos dieser Tabelle an die Stao, damit diese mit den Delta-Schnappschüssen auch löschen können.
DBStateValues	Speichert den Status was geloggt werden soll, um bei der Übernahme der Daten ausgewählte Trigger auf den Tabellen kurzfristig ab-/einschalten zu können.

Tabelle 6: neue Tabellen

Es wird eine neue Tabelle zur (automatischen) Speicherung der geänderten Vorlagen-Elemente (Ordner, Merkmale, Parameter usw.) erstellt (→ " LocalModifiedElements"). Diese Tabelle beinhaltet die für die Notification-Publikation notwendigen Informationen. Hier wird auch der Status der Änderungen festgehalten (offen, akzeptiert, zurückgewiesen). Jedes geänderte Element besitzt nur genau einen Eintrag in dieser Tabelle. Nicht geänderte Elemente sind nicht eingetragen.→ Ziel: der Status der Multiserver-Funktionalität soll so weit wie möglich von den "normalen" Datentabellen entkoppelt werden.

Erstellung der Tabellen:

Über das SQL Skript: CreateAutoLog4TemplatesDisableChangeLog.sql

Tabelle LocalModifiedElements

```
if not exists (select * from dbo.sysobjects where id =
object_id(N'[dbo].[LocalModifiedElements]') and OBJECTPROPERTY(id,
N'IsUserTable') = 1)
create table [dbo].[LocalModifiedElements] (
      [Table_ID] [int],
      [Element_ID] [uniqueidentifier],
      [Modified_UTC] [datetime],
      [ModifyingUser] [nvarchar] (50) NULL,
      [Snapshot_UTC] [datetime] NULL,
      [Removed] [bit] NULL,
      [ModAccepted] [bit] NULL constraint
DF_LocalModifiedElements_ModAccepted default 0,
      [ModRejected] [bit] NULL constraint
DF_LocalModifiedElements_ModRejected default 0
)
go
```

Alle (Vorlagen-)Tabellen, welche am Standort bearbeitet werden dürfen, müssen mit Triggern versehen, werden, damit alle Änderungen automatisch in "LocalModifiedElements" nachgeführt werden.

Erstellung der Trigger:

Über das SQL Skript: TemplateTriggersInsDel.sql

Und TemplateTriggersUpdate.sql → für Logbucheinträge für jede geänderte Spalte

Beispiel auf der Tabelle Parameter für Eintrag in LocalModifiedElements

```
if exists (select * from dbo.sysobjects where id =
object_id('dbo.trParameter_Modification'))
drop trigger dbo.trParameter_Modification
go
create trigger dbo.trParameter_Modification
on Parameter
for update, insert, delete as
  if @@ROWCOUNT=0 return
  if not exists(select * from deleted) or
      not (update(Timestamp_UTC) and not update(ID)) and not
update(LastModify)
        and not ([dbo].fnDbStateGetTemplateLoggingDisabled() <> 0)
  begin
    declare @ModifiedTableID int
    declare @TableName nvarchar (50)
    -------------------------------------------
```

```
set @ModifiedTableID = 1
set @TableName = 'Parameter'
-------------------------------------------

declare @Action char
declare @RowIDs table([ID] uniqueidentifier)
declare @User nvarchar (50)
declare @Removed bit

set @User = dbo.fnGetTemplateLockingUser()
if @User is null or @User = ''
  set @User = dbo.fnGetMachineParameterLockingUser()
if exists(select * from inserted)
begin
  insert into @RowIDs ([ID]) select [ID] from inserted
  set @Removed = 0
  if exists(select * from deleted)
    set @Action = 'U'     -- update
  else
    set @Action = 'I'     -- insert
end
else
begin
  insert into @RowIDs ([ID]) select [ID] from deleted
  set @Action = 'D'       -- delete
  set @Removed = 1
end

if @Action = 'U'
begin
  select * into #tmp_inserted from inserted
  select * into #tmp_deleted from deleted
  declare     @TablesAreEqual bit
  exec dbo.spCompareTableContentsForEquality
  @TableA = '#tmp_inserted',
  @TableB = '#tmp_deleted',
  @ColumnListSrcTable = @TableName,
  @TablesAreEqual = @TablesAreEqual OUTPUT
  if @TablesAreEqual <> 0
    return
end
```

```
    update [dbo].[LocalModifiedElements] set [Modified_UTC] = GETUTCDATE(),
[ModifyingUser] = @User, [Removed] = @Removed where
                  ([Table_ID] = @ModifiedTableID) and ([Element_ID] in
(select [ID] from @RowIDs))

    insert into [dbo].[LocalModifiedElements] ([Table_ID], [Element_ID],
[Modified_UTC], [ModifyingUser], [Snapshot_UTC], [Removed])
                  select @ModifiedTableID, [ID], GETUTCDATE(), @User, null,
@Removed from @RowIDs where
                  ([ID] not in (select [Element_ID] from
[dbo].[LocalModifiedElements] where [Table_ID] = @ModifiedTableID))

    if (@Action <> 'U') and not
([dbo].fnDbStateGetTemplateChangeLogDisabled() <> 0)
      insert into [dbo].[AutoLog_Templates] ([TimeStamp], [TableName],
[Element_ID], [Action], [UserName], [Origin_Loc_ID])
                  select GETDATE(), @TableName, [ID], @Action, @User, null
from @RowIDs
  end
go
```

Das automatische Logging muss deaktivierbar sein (z.B. für Vorlagen-Import oder während der Übernahme der "Master Schnappschüsse").

Erstellung des LoggingStatus:

Über das SQL Skript: „CreateAutoLog4TemplatesDisableChangeLog.sql“

DBStateValues

```
if not exists (select * from dbo.sysobjects where id =
object_id(N'[dbo].[DbStateValues]') and OBJECTPROPERTY(id, N'IsUserTable')
= 1)
create table [dbo].[DbStateValues] (
      [ID] [int] NOT NULL,
      [Name] [nvarchar] (50) NULL,
      [Value] [varbinary] NULL,
      [LastModify] [datetime] NULL
)
go
```

Stored Procedure um das Logging zu de-/aktivieren:

```
if exists (select * from dbo.sysobjects where id =
object_id('dbo.spDbStateSetTemplateLoggingDisabled'))
  drop procedure dbo.spDbStateSetTemplateLoggingDisabled
go
create procedure dbo.spDbStateSetTemplateLoggingDisabled(@NewState bit)
as
  update [dbo].[DbStateValues] set [Value] = @NewState where ([ID] = 1)
go
```

Funktion um den derzeitigen Logging Status abzufragen

```
if exists (select * from dbo.sysobjects where id =
object_id('dbo.fnDbStateGetTemplateLoggingDisabled'))
  drop function dbo.fnDbStateGetTemplateLoggingDisabled
go
create function dbo.fnDbStateGetTemplateLoggingDisabled()
returns bit
begin
  return (select [Value] from [dbo].[DbStateValues] where ([ID] = 1))
end
go
```

Die Tabellen der Vorlagen-Elemente werden generell um Zeitstempel-Spalten zur Konfliktkennung/-Lösung erweitert. Der Zeitstempel muss unabhängig von der lokalen Zeitzone sein (UTC)

Erweitern der Vorlagen Tabellen:

Über das SQL Skript: „AlterTables.sql“

Beispiel:

```
if not exists (select * from dbo.syscolumns where (id =
object_id(N'[dbo].[Parameter]')) and (name = 'Timestamp_UTC'))
  alter table [dbo].[Parameter] ADD
  [Timestamp_UTC] [datetime] NULL,
  [LastModify] [datetime] NULL
```

Die zu synchronisierenden Beziehungstabellen müssen mit eigenen (für die eigentliche Funktion an sich überflüssigen) GUID Primärschlüsseln erweitert werden.

Über das SQL Skript: „AlterTables.sql“

Beispiel:

```
if not exists (select * from dbo.syscolumns where (id =
object_id(N'[dbo].[Bez_RezPrm]')) and (name = 'LastModify'))
  alter table [dbo].[Bez_RezPrm] ADD
  [Version] [int] NULL,
  [ID] [uniqueidentifier] NULL,
  [LastModify] [datetime] NULL
```

Zur Beschleunigung der Zuordnung von Ordnern zu Rezepten wird eine neue Beziehungs-Tabelle Ordner ←→ Rezept erstellt.

Über das SQL Skript: „CreateLocalLocExportDB.sql“

```
CREATE TABLE [dbo].[Ordner_Rcp] (
      [ID] [uniqueidentifier] NOT NULL ,
      [Name] [nvarchar] (50) COLLATE SQL_Latin1_General_CP1_CI_AS NOT NULL
,
      [Parent_ID] [uniqueidentifier] NULL ,
      [Referenz_ID] [uniqueidentifier] NULL ,
      [Berechtigung] [int] NULL ,
      [Typ] [int] NULL ,
      [Reihenfolge] [int] NULL ,
      [Obligatorisch] [bit] NOT NULL ,
      [DynInfoParameter_ID] [uniqueidentifier] NULL ,
      [Bemerkung] [nvarchar] (100) COLLATE SQL_Latin1_General_CP1_CI_AS
NULL ,
      [XML_Element] [nvarchar] (100) COLLATE SQL_Latin1_General_CP1_CI_AS
NULL ,
      [Ursprung_Stao_ID] [int] NULL ,
      [Timestamp_UTC] [datetime] NULL ,
      [LastModify] [datetime] NULL
) ON [PRIMARY]
GO
```

Die Logbücher müssen um ein 'Ursprungs-Standort'-Feld erweitert werden.

Über das SQL Skript: „AlterTables.sql“

Beispiel:

```
if not exists (select * from dbo.syscolumns where (id =
object_id(N'[dbo].[Logbuch]')) and (name = 'Origin_Loc_ID'))
  alter table [dbo].[Logbuch] ADD
  [Origin_Loc_ID] [int] NULL
```

6.2 Konfigurationshinweise für die Replikation

Um die Replikation einzurichten müssen zuerst die zu publizierenden Export Datenbanken am HS und am STAO angelegt werden.

Am **HS** sind das die RZNW-MainExport für die Snapshots zu den STAOs

Über das SQL Skript: „CreateMainExportDB.sql“

Und die RZNW-LocImport für jeden STAO, der seine Snapshots zum HS publiziert

Über das SQL Skript: „CreateMainLocImport_XxxDB.sql“

Am **STAO** für den Snapshot zum HS

Über das SQL Skript: „CreateLocalLocExportDB.sql“

Und die Datenbank, die die Snapshots vom HS empfängt

Über das SQL Skript: „CreateLocalMainImportDB.sql“

Alle notwendigen Datenbanken am Hauptserver:

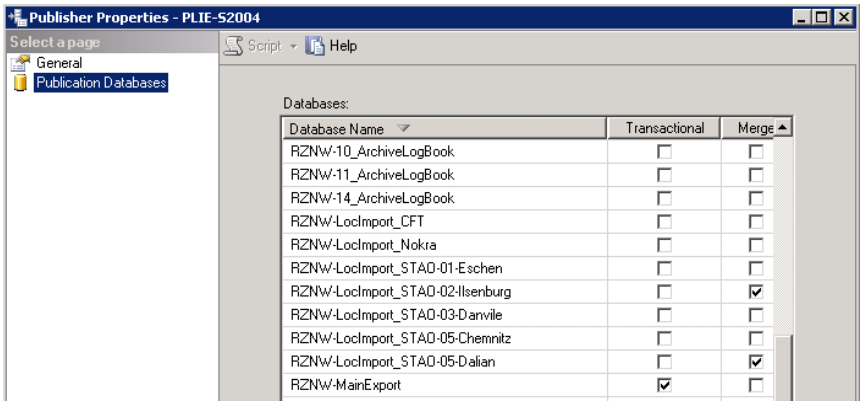


Abb. 30: alle am HS erstellten LocImport und MainExport Datenbanken

STAO:

Die Replikationseinstellungen für den Publisher und Distributor sind im SQL Skript

„CreateReplicationPublicationsLocationExport.sql” enthalten

Es werden die Vorlagen, Rezeptur, Logbuch und Maschinenverwaltungs-Publikationen der LocExport Datenbanken mit den korrekten Artikeln erstellt. Am HS müssen die Subscriptions auf folgende Publikationen richtig konfiguriert werden.

für die Vorlagen	→ Location-Templates
die Rezepte	→ Location-Recipes
die Logbücher	→ Location-Logs
Die Maschinenverwaltung	→ Location-MachineAdmin

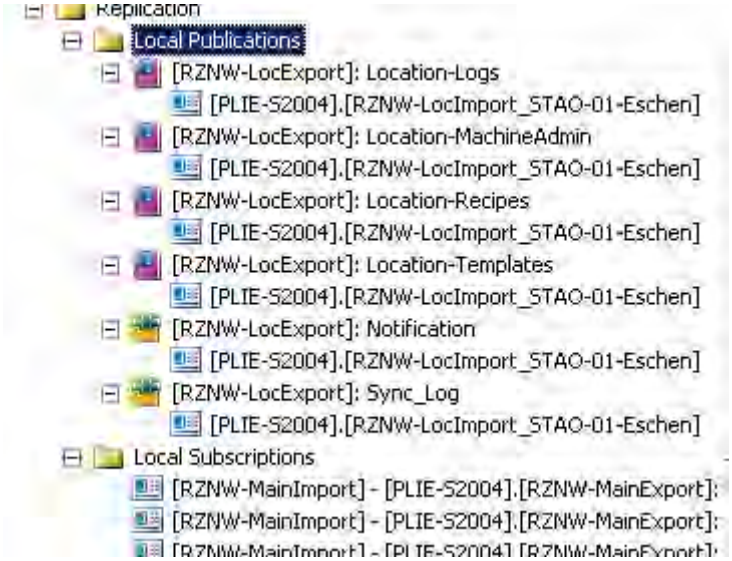


Abb. 31: STAO alle Publikationen zum HS und die Subscriptions auf HS Publikationen

HS:

Die Replikationseinstellungen für den Publisher und Distributor sind im SQL Skript

„CreateReplicationPublicationsMainExport.sql" enthalten.

Es werden die Publikationen der MainExport-Datenbank mit den korrekten Artikeln erstellt. Am STAO müssen die Subscriptions auf diese Publikationen richtig konfiguriert werden.

Die Replikation auf den MS SQL Servern gerade bei unterschiedlichen Versionen ist sehr komplex und weicht im Detail erheblich voneinander ab. Dazu muss unbedingt auf die MSDN und die Produkt Dokumentation verwiesen werden.

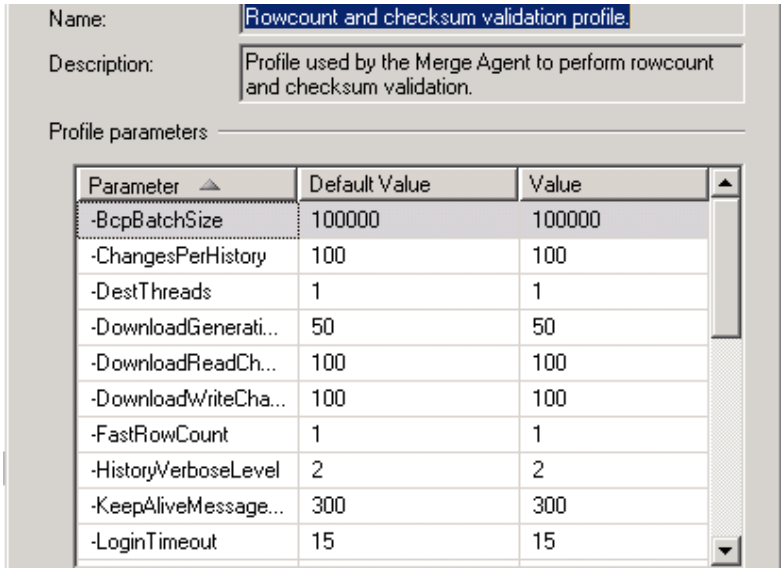


Abb. 32: Job Profile eines Agenten: Feintuning at its best

6.3 Erzeugen von RZNW-Schnappschüssen am Stao

6.3.1 Export-DB

Die für den Hauptserver bestimmten Daten werden nicht direkt aus der RZNW-DB heraus publiziert, sondern als Schnappschuss in eine eigens dafür vorgesehene Export-DB abgelegt ("RZNW-LocExport"). In dieser einen DB sind die Daten für alle Publikationen des Standortservers enthalten. Ebenso enthält sie die für den Datentransfer benötigten Funktionen und Prozeduren.
Diese Trennung hat den Vorteil, dass aus der RZNW-DB heraus zu einem ganz bestimmten, kontrollierten Zeitpunkt ein Schnappschuss der aktuellen Daten erstellt werden kann und dieser dann in der Export-DB zur Replikation "geparkt" wird.

6.3.2 Ablauf der Datenübertragung

Um die Daten per Replikation vom Standortserver zum Hauptserver zu übertragen sind der Reihe nach folgende Schritte nötig:

1) Erzeugen des aktuellen RZNW-Datenschnappschusses aus der STAO Datenbank in die Export-DB. Für jede der Publikationen kann unabhängig ein eigener Schnappschuss erstellt werden.
2) Replikations-Snaphot-Agent für jede Publikation mit einem neuen RZNW-Schnappschuss ausführen. Die Snapshot-Agents erstellen aus der Export-DB die binären Snapshots zur Transferierung zum HS.
3) Distribution Agent des STAO ausführen. Dieser überträgt den binären Schnappschuss per Netzwerkverbindung in die Import-DB des HS.
4) Merge-Agent für die Sync_Log Publikation ausführen.

6.3.3 Schnappschuss der RZNW-DB erzeugen

Zum Transferieren eines aktuellen Schnappschusses der RZNW-DB in die Export-DB existiert für jede Publikation eine unabhängige gespeicherte Prozedur:

Publikation	**Gespeicherte Prozedur**
Vorlagen	spMsMakeLocExportTemplatesSnaphot Parameter: keine
Rezepte	spMsMakeLocExportRecipesSnaphot Parameter: Start-Zeitpunkt für Snapshot

Maschinenverwaltung	spMsMakeLocExportMachAdminSnaphot Parameter: Start-Zeitpunkt für Snapshot
Logbücher	spMsMakeLocExportLogSnaphot Parameter: Start-Zeitpunkt für Snapshot

Tabelle 7: Schnappschussgenerierung StoredProcedures am STAO

Die Schnappschüsse der einzelnen Publikationen sind unabhängig voneinander und können einzeln ausgeführt werden.

Beispiel der Ausführung:

```
EXEC spMsMakeLocExportTemplatesSnaphot
EXEC spMsMakeLocExportRecipesSnaphot '2011-01-01 00:00:00'
EXEC spMsMakeLocExportLogSnaphot '2011-01-01 00:00:00'
EXEC spMsMakeLocExportMachAdminSnaphot '2011-01-01 00:00:00'
```

Der Start-Zeitpunkt des zu erstellenden Schnappschusses wird dabei üblicherweise direkt der Sync_Log Tabelle entnommen und ist der Schnappschuss-Zeitstempel des zuletzt am Hauptserver übernommenen Schnappschusses des Standorts. Zur einfachen Abfrage dieses Zeitstempels ist in der Export-DB eine Funktion hinterlegt:

```
exec dbo.fnMsGetLatestAppliedPubSnaphotTimestamp
```

Sie übernimmt als Argument eine Kennung für den gewünschten Schnappschuss:

'R' →Rezepturen

'M' →Maschinenverwaltung

'L' → Logbücher

Die automatisierte Ausführung der obigen Aufrufe sieht demnach wie folgt aus:

```
declare @TimeStampInfo datetime
exec dbo.spMsMakeLocExportTemplatesSnaphot
set @TimeStampInfo = (select [RZNW-
LocExport].dbo.fnMsGetLatestAppliedPubSnaphotTimestamp('R'))
exec dbo.spMsMakeLocExportRecipesSnaphot @TimeStampInfo

set @TimeStampInfo = (select [RZNW-
LocExport].dbo.fnMsGetLatestAppliedPubSnaphotTimestamp('L'))
exec dbo.spMsMakeLocExportLogSnaphot @TimeStampInfo

set @TimeStampInfo = (select [RZNW-
LocExport].dbo.fnMsGetLatestAppliedPubSnaphotTimestamp('M'))
exec dbo.spMsMakeLocExportMachAdminSnaphot @TimeStampInfo
```

6.3.4 Snapshotgenerierung über Agenten per SQL starten

Für die Auslösung per T-SQL-Kommando existieren in der Export-DB für jede der Publikationen gesonderte gespeicherte Prozeduren (alle ohne Parameter):

Publikation	Gespeicherte Prozedur
Vorlagen	dbo.spMsStartSnapshotAgent4Templates
Rezepte	dbo.spMsStartSnapshotAgent4Recipes
Maschinenverwaltung	dbo.spMsStartSnapshotAgent4MachAdmin
Logbücher	dbo.spMsStartSnapshotAgent4Logs

Tabelle 8: Snapshot Agenten per StoredProcedures am STAO ausführen

Die Agenten der einzelnen Publikationen sind unabhängig voneinander und können einzeln ausgeführt werden.

Beispiel der (kompletten) Ausführung:

```
EXEC [RZNW-LocExport].dbo.spMsStartSnapshotAgent4Templates
EXEC [RZNW-LocExport].dbo.spMsStartSnapshotAgent4Recipes
EXEC [RZNW-LocExport].dbo.spMsStartSnapshotAgent4MachAdmin
EXEC [RZNW-LocExport].dbo.spMsStartSnapshotAgent4Logs
```

Die Snapshot-Agents benötigen zur Ausführung einige Zeit und laufen in einem eigenen Thread, so dass, falls Distributions- und Merge-Agenten gleich anschliessend im selben Skript gestartet werden sollen, zuvor eine Wartezeit eingelegt werden muss. Ansonsten findet der Distributions-Agent den neuen Schnappschuss noch nicht vor und die Daten werden dann folglich auch nicht zum HS übertragen.

Der Wartebefehl könnte so aussehen:

```
WAITFOR DELAY '00:05:00'
```

Dieses Kommando lässt die weitere Batch-Ausführung 5 Minuten warten.

6.3.5 Distribution Agenten per SQL starten

Beispiel der Ausführung:

```
EXEC [RZNW-LocExport].dbo.spMsStartDistributionAgent4LocExport
EXEC [RZNW-LocExport].dbo.spMsStartSyncLogMergeAgent
EXEC [RZNW-LocExport].dbo.spMsStartNotificationMergeAgent
```

6.4 Handshake Stao → HS

6.4.1 Sync_Log

Die Tabelle Sync_Log dient dem Austausch von Status-Meldungen zwischen Hauptserver und Standort-Server bezüglich der vom Standortserver bereitgestellten Schnappschüsse. Die Tabelle befindet sich in der Stao-DB "RZNW-LocExport" und wird per Merge-Replikation zum Hauptserver repliziert.

Spalte	Beschreibung
ID	Eindeutige GUID zur Identifikation des Eintrags
TimeStamp_UTC	UTC-Zeitstempel der Erzeugung des Eintrags
ServerName	Name des eintragenden SQL Servers
Publication	Betroffene Publikation: 'T' → Vorlagen (Tempates) 'R' → Rezepte (Recipes) 'M' → Maschinenverwaltung 'L' → Logbücher
Event	Ereigniscode 0 → ungültig 1 → Schnappschuss erstellt 2 → Schnappschuss erfolgreich übernommen 3 → Übernahme ist fehlgeschlagen
EventInfo	Optionale Zusatzinformation (Text) zum Ereignis, z.B. Fehlermeldung/Code im Fehlerfall
SyncTimeStampInfo	Optionale Zeitstempel-Info (Lokalzeit) zu Ereignis (z.B. Zeitstempel des neuesten Log-Eintrags bei Übernahme des Log-Snapshots) Stao → HS: Zeitstempel des Snapshots HS → Stao: Zeitstempel des übernommenen Snapshots ('gespiegelt' von auslösendem Ereignis)
Ack	Bestätigung (Acknowledge) des Eintrages durch die jeweils andere Seite. Durch Setzen dieses Bits signalisiert z.B. der HS, dass er einen neuen Schnappschuss zur Kenntnis genommen hat. Durch diesen Mechanismus wird die doppelte Verarbeitung eines Ereignisses vermieden und die Rückverfolgbarkeit des Datenflusses verbessert.
AckTime_UTC	UTC-Zeitstempel der Bestätigung.
AckInfo	Optionale Zusatzinformationen (Text) der Bestätigung
AckServerName	Name des bestätigenden SQL Servers

Tabelle 9: Attributbeduetung der Sync_Log Tabelle

Einträge

Zum Vornehmen und Bearbeiten von Einträgen werden Stored Procedures zur Verfügung gestellt. Diese sind ebenfalls in der DB "RZNW-LocExport" abgelegt und werden in die Gegen-DB auf dem Hauptserver repliziert.

Ablauf für Schnappschüsse mit Datenübernahme am Hauptserver (Publikationen 'R', 'M' und 'L')

a. Der Standortserver generiert einen neuen Schnappschuss. Den aktuellen Zeitstempel des Schnappschusses (Zeitpunkt kurz vor der Erstellung) trägt er in die entsprechende Zeile der Tabelle "SnapshotInfo" (in der RZNW-LocExport DB) ein.

b. Der Standortserver erzeugt einen neuen Eintrag im Sync_Log mit Event "1". Als "SyncTimeStampInfo" trägt er exakt den gleichen Zeitpunkt ein wie zuvor in die Tabelle "SnapshotInfo".

c. Die Snapshot-Publikationen werden erstellt und über den Distributor am STAO auf den HS repliziert. Ebenso das Sync_Log per Merge-Publikation, wobei dieses zeitversetzt (vor oder nach den Snapshot Publikationen) übertragen werden kann.

d. Der HS "bemerkt" den neuen Snapshot vom Standort per zyklischer Prüfung der gemergten Sync_Log in der RZNW-LocImportxxx Datenbank. Der neue Schnappschuss ist erst dann wirklich gültig (vorhanden), wenn die entsprechende Publikations-Zeile der Tabelle "SnapshotInfo" den selben neuen Zeitstempel trägt, wie er im neuen Sync_Log-Eintrag vermerkt ist. So wird sichergestellt, dass auch wirklich der korrekte Schnappschuss vorliegt.

e. Am Hauptserver können nun die Daten aus der RZNW-LocImportxxx Datenbank in die Haupt-DB übernommen werden.

f. Bei erfolgreicher Übernahme der Daten:

   Der Hauptserver erzeugt zuerst einen neuen Eintrag im Sync_Log mit Event "2", dann bestätigt er den gerade bearbeiteten Eintrag des Standortservers per Acknowledge.

   Bei fehlgeschlagener Übernahme der Daten: Der Hauptserver erzeugt einen neuen Eintrag im Sync_Log mit Event "3", und bestätigt den gerade bearbeiteten Eintrag des Standortservers nicht per Acknowledge. Der Eintrag (Auftrag) bleibt somit offen.

6.4.2 Notification

Die sog. Notification dient dem Austausch von Änderungs- bzw. Status-Meldungen zwischen Standortserver und Hauptserver bezüglich der am Standortserver geänderten, hinzugefügten oder gelöschten Elemente der Vorlagen und Hilfsdaten.

Als Notification wird die Tabelle "LocalModifiedElements" der Export-DB des Standortservers verwendet ("RZNW-LocExport"). Die Tabelle wird per Merge-Replikation zum Hauptserver repliziert.

Die Daten für die LocalModifiedElements-Tabelle stammen ursprünglich aus der gleichnamigen Tabelle der Standort-DB. In diese werden per Trigger automatisch alle Änderungen an Vorlagen bzw. Hilfsdaten-Elementen vermerkt.

Spalte	**Beschreibung**
Table_ID	Code (ID) für Tabelle, auf welche sich der Eintrag bezieht.
Element_ID	Eindeutige ID des Elements, auf welches sich der Eintrag bezieht (z.B. Ordner-ID).
Modified_UTC	UTC-Zeitstempel der Änderung Das Festhalten des Zeitpunkts per UTC ermöglicht am Hauptserver im Falle von Mehrfachänderungen an verschiedenen Standorten (Konflikte) die korrekte chronologische Reihung der Änderungen unabhängig von der jeweiligen Lokalzeit.
ModifyingUser	Name des für die Änderung verantwortlichen Benutzers. Hierzu wird im Trigger der Name des Benutzers ermittelt, welcher gerade die Vorlagen oder Hilfsdaten gesperrt hält (editiert).
Snapshot_UTC	UTC-Zeitstempel der Übernahme des Eintrags in den Schnappschuss des Standortservers (erstmalige Replikation)
Removed	Bit (boolean), welches gesetzt (1) wird, wenn das Element, auf welches sich der Eintrag bezieht, lokal gelöscht wurde. Das Problem hierbei ist, dass dieses Element lokal gar nicht mehr vorhanden ist, am Hauptserver hingegen schon. Dort kann in der entsprechenden Ansicht für das (dort noch vorhandene) Element ein "Löschan-

	trag" dargestellt werden.
ModAccepted	Rückmeldung vom Hauptserver, dass die lokal vorgenommene Änderung des betroffenen Elements akzeptiert wurde.
ModRejected	Rückmeldung vom Hauptserver, dass die lokal vorgenommene Änderung des betroffenen Elements abgelehnt wurde.

Tabelle 10: Attributbeduetung der LocalModifiedElements Tabelle der Notification Publikation

Ablauf für Schnappschüsse mit Datenübernahme am Hauptserver (Publikation 'T')

1) Der Standortserver generiert einen neuen Schnappschuss. Aus der Standort-DB werden jene Elemente kopiert, welche lokal modifiziert oder neu erstellt worden sind. Dies sind alle Elemente, welche in der Tabelle "LocalModifiedElements" einen Eintrag besitzen. Hierbei müssen aber folgende Fälle berücksichtig werden:

   - Element im vorausgehenden Schnappschuss nicht enthalten → Element wird kopiert (einfachster Fall)
   - Element bereits im vorausgehendem Schnappschuss, "ModAccepted" und "ModRejected" sind beide 0 → Es handelt sich um eine alte, am Hauptserver noch nicht bearbeitete Änderung/Erweiterung. Das Element im vorausgegangenen Schnappschuss (alte Version) wird ersetzt. Auf diese Weise ist sichergestellt, dass:

     a). Eine Änderung nicht "vergessen" geht

     b). Bei der verzögerten Bearbeitung am HS die aktuelle Version zur Verfügung steht.
   - Element bereits im vorausgehendem Schnappschuss, "ModAccepted" oder "ModRejected" ungleich 0 → (vorausgegangene Änderung des Elements wurde zwischenzeitlich am Hauptserver bearbeitet. Das Resultat wird bei Übernahme der Hauptserver-Vorlagen berücksichtigt und in die Standort-DB eingepflegt. Das Element wird nicht in den neuen Schnappschuss übernommen.

     Ob sich das aktuelle lokale Element von jenem im letzten Schnappschuss unterscheidet, kann aufgrund des "LastModify"-Feldes entschieden werden.

2) Der STAO erzeugt einen neuen Eintrag im Sync_Log mit Event "1", Publikation 'T'
3) Der Schnappschuss wird auf den HS repliziert
4) Der HS "bemerkt" den neuen Schnappschuss vom Standort per zyklischer Prüfung des Sync_Log.

5) Der Admin am HS bearbeitet die von den STAO offenen Änderungen. Er muss nicht alle offenen Änderungen bearbeiten, sondern kann auch nur einen Teil davon berücksichtigen. Resultat dieser Bearbeitung ist entweder das Akzeptieren der Änderung ("ModAccepted" wird 1) oder das Zurückweisen derselben ("ModRejected" wird 1). Auf jeden Fall wird dies im entsprechenden Feld des Element-Eintrags in der RZNW-LocImport-DB (Tabelle "LocalModifiedElements") des Standorts auf dem HS eingetragen und beim nächsten Replikationszyklus (Merge) an den Standort zurückübertragen.

6) Am Standort werden nun alle Einträge der Notification abgearbeitet, welche entweder akzeptiert oder zurückgewiesen wurden:

6.5 Übernahme der Standort-Daten am Hauptserver

6.5.1 Zugriff auf die Haupt-DB

Die Daten der verschiedenen Standorte liegen (nach der Replikation) in einer jeweils eigenen Import-Datenbank auf dem Hauptserver für die Übernahme/Bearbeitung bereit.

6.5.2 Stored Procedures

Zur Übernahme der Publikationen 'R', 'M' und 'L' aus den Standorten in den Hauptserver steht innerhalb jeder Import-DB pro Publikation eine gespeicherte Prozedur zur Verfügung:

Publikation	Gespeicherte Prozedur
Rezepte	spMsApplyLocExportRecipeSnaphot Parameter: keine
Maschinenverwaltung	spMsApplyLocExportMachAdminSnaphot Parameter: keine
Logbücher	spMsApplyLocExportLogSnaphot Parameter: keine

Tabelle 11: Skripte für die Datenübernahme-von der Import DB in die HS DB

Die Prozeduren fügen die neuen Daten in die entsprechenden Tabellen ein und ändern die bereits vorhandenen auf den neuen Stand. Nach erfolgreicher Übernahme vermerken sie den Zeitstempel des gerade bearbeiteten Schnappschusses im Sync_Log, so dass der Standortserver bei der Erstellung des nächsten Schnappschusses die nun gerade eingefügten Daten nicht erneut zur Verfügung stellt. Auf diese Weise kann die pro Zyklus zu transferierende Datenmenge klein gehalten werden.

6.5.3 Effektive Übernahme

Die effektive Übernahme der neuen Daten geschieht durch den Aufruf der oben beschriebenen Prozeduren. Die Prozeduren erkennen dabei automatisch, ob überhaupt neue Daten zur Verfügung stehen oder nicht. Falls nicht, beenden sie sich sofort wieder ohne weitere Aktionen.

Der Aufruf kann auch zyklisch erfolgen, und zwar im einfachsten Fall über einen per Intervall gesteuerten Job des SQL Server Agents. Die einzelnen Publikationen können dabei flexibel entweder zusammen in einem Job oder aber zu unterschiedlichen Intervallen über individuelle Jobs bearbeitet werden. Der Aufruf sieht bei einem gemeinsamen Job so aus:

```
use RZNW_LocImportxxx
exec dbo.spMsApplyLocExportLogSnaphot
exec dbo.spMsApplyLocExportRecipeSnaphot
exec dbo.spMsApplyLocExportMachAdminSnaphot
```

Die Prozeduren müssen für jede Import-Datenbank separat ausgeführt werden, lassen sich aber auch über entsprechende "USE"-Anweisungen in einem einzigen Job zusammenfassen.

6.5.4 Geänderte Vorlagen

Die an den Standorten vorgenommenen Vorlagen-Änderungen können am Hauptserver im RZNW-Multiserver-Frontend bearbeitet werden.

Vor der Bearbeitung können bereits gewisse automatische Filter angewendet werden:

- Änderungen von Elementen, welche am Standort gelöscht wurden ("Removed"-Bit gesetzt), am Hauptserver aber nicht (mehr) vorhanden sind, werden auto-akzeptiert. Es handelt sich dabei entweder um Elemente, die lokal angelegt und gleich wieder gelöscht wurden, oder um solche, welche mittlerweile auch vom Server entfernt wurden und damit die Löschung vom Standortserver implizit ebenfalls legitimiert ist.
- Gemeldete Änderungen, bei denen sich das Standort-Element nicht vom HS-Element unterscheidet (bis auf die Zeitstempel alle Felder gleich) werden auto-akzeptiert, da die Änderung am Standort entweder zwischenzeitlich wieder rückgängig gemacht wurde oder dieselbe Änderung am Hauptserver ebenfalls vorgenommen worden ist. Der Änderungsauftrag ist somit obsolet und kann auto-akzeptiert werden.

6.6 Erzeugen von RZNW-Schnappschüssen am Hauptserver

6.6.1 Export-DB

Die für die Standortserver bestimmten Daten werden nicht direkt aus der RZNW-DB heraus publiziert, sondern als Schnappschuss in eine eigens dafür vorgesehene Export-DB abgelegt ("**RZNW-MainExport**"). In dieser einen DB sind die Daten für alle Publikationen des Hauptservers enthalten. Ebenso sind in ihr auch die für den Datentransfer benötigten Funktionen und Prozeduren gespeichert.

Diese Trennung hat den Vorteil, dass aus der RZNW-DB heraus zu einem ganz bestimmten, kontrollierten Zeitpunkt ein Schnappschuss der aktuellen Daten erstellt werden kann und dieser dann in der Export-DB zur Replikation "geparkt" wird.

6.7 Übernahme der Hauptserver-Daten am Standort

Der HS publiziert den aktuellen (freigegebenen) Schnappschuss seiner RZNW-Daten in seiner Export-Datenbank und stellt diese den Standorten über die oben beschriebenen Publikationen zur Verfügung.

Publikation	Gespeicherte Prozedur
Vorlagen	dbo.spMsMakeMainExportTemplatesSnaphot
Rezepte	dbo.spMsMakeMainExportRecipesSnaphot
Maschinenverwaltung	dbo.spMsMakeMainExportMachAdminSnaphot

Tabelle 12: Schnappschusserstellungsscript am HS

Die Stored Procedures der einzelnen Publikationen sind unabhängig voneinander und können einzeln ausgeführt werden.
Beispiel der (kompletten) Ausführung:

```
EXEC [RZNW-MainExport].dbo.spMsMakeMainExportTemplatesSnaphot
EXEC [RZNW-MainExport].dbo.spMsMakeMainExportRecipesSnaphot
EXEC [RZNW-MainExport].dbo.spMsMakeMainExportMachAdminSnaphot
```

Die STAOs greifen die Daten per Pull-Subscription (via Snapshot Replikation) vom Hauptserver ab. Die Pull-Subscription dient der Entlastung des HS und dem individuellen Timing für jeden STAO. Das Timing kann so an die spezifischen Gegebenheiten angepasst werden (z.B. an die zur Verfügung stehende begrenzte Bandbreite.
Der Distribution-Agent läuft also folglich auf dem STAO und kann dort so konfiguriert werden, dass er automatisch zyklisch ausgeführt wird.

Die Ausführung kann auch manuell per Stored Procedure ausgelöst werden:

```
EXEC [RZNW-STAOxxx].dbo.spMsStartDistributionAgent4MainExport
```

Die Replikation transferiert die aktuellen Daten vom HS in die Import-DB des STAOs ("**RZNW-MainImport**"). Von dort werden sie per Stored Procedures in die Standort-DB übernommen.

6.7.1 Vorlagen

Von den Vorlagen und Hilfsdaten werden jene Elemente übernommen, welche nicht am jeweiligen Standort geändert und somit als provisorisch gekennzeichnet sind. Diese definitiven HS-Vorlagen werden ohne weitere Prüfung übernommen.

Lokal geänderte Elemente werden gemäss den Notification-Infos geprüft und entweder als definitiv übernommen oder aber

- als abgelehnt zum manuellen Löschen markiert (abgelehnte neue Elemente)
- wieder hergestellt (lokal gelöschte Elemente)
- in den aktuellen HS-Zustand versetzt (abgelehnte Änderungen)

6.7.2 Rezepte

Bei den Rezepten wird je nach Standort des Rezepts individuell vorgegangen:

- Die Rezepte der anderen Standorte werden ungeprüft übernommen, da sie lokal ohnehin nicht geändert werden dürfen
- Rezepte des eigenen Standorts, welche in der Standort-DB noch nicht vorhanden sind, werden ohne weitere Prüfung übernommen. Es handelt sich um neue, am Hauptstandort angelegte Rezepte bei ihrer erstmaligen Übertragung.
- Rezepte des eigenen Standorts, welche in der Standort-DB schon vorhanden sind, werden ignoriert. Da am Hauptstandort (und den übrigen sowieso) daran nichts geändert werden darf, sind lokal ohnehin die aktuellen Daten vorhanden.

6.7.3 Maschinenverwaltung

Die Vorgehensweise bei der Übernahme des Schnappschusses der Maschinenverwaltung wird ebenfalls nach Standort unterschieden:

- Standort- und Geschäftsbereichsdaten werden ungeprüft übernommen. Sie dürfen nur am Hauptserver geändert werden.
- Linien und Maschinen-Daten (inkl. Anlagenkarten, Maschinenkarten, Maschineninfos, Maschinen-Operationen) der anderen Standorte werden ungeprüft übernommen.

- Linien und Maschinen (+ deren Daten) des eigenen Standorts, welche in der lokalen Standort-DB noch nicht vorhanden sind, werden ohne weitere Prüfung übernommen. Es handelt sich um neue, am Hauptstandort angelegte Linien/Maschinen bei ihrer erstmaligen Übertragung.
- Linien und Maschinen des eigenen Standorts, welche in der Standort-DB schon vorhanden sind, werden ignoriert. Da am Hauptstandort (und den übrigen sowieso) daran nichts geändert werden darf, sind lokal ohnehin die aktuellen Daten vorhanden.

6.7.4 Stored Procedures

Zur Übernahme der aktuellen Daten vom Hauptserver steht in der Standort-DB pro Publikation eine gespeicherte Prozedur zur Verfügung:

Publikation	Gespeicherte Prozedur
Vorlagen	spMsApplyMainExportTemplateSnaphot
Rezepte	spMsApplyMainExportRecipeSnaphot
Maschinenverwaltung	spMsApplyMainExportMachAdminSnaphot

Tabelle 13: Schnappschussanwendungscript am STAO

Die Prozeduren fügen die neuen Daten in die entsprechenden Tabellen ein und ändern die bereits vorhandenen auf den neuen Stand.

6.7.5 Effektive Übernahme der Daten vom HS in die STAO Datenbank

Die effektive Übernahme der neuen Daten geschieht durch den Aufruf der oben beschriebenen Prozeduren. Die Prozeduren erkennen dabei automatisch, ob überhaupt neue Daten zur Verfügung stehen oder nicht. Falls nicht, beenden sie sich sofort wieder ohne weitere Aktionen.

Der Aufruf kann auch zyklisch erfolgen, und zwar im einfachsten Fall über einen per Intervall gesteuerten Job des SQL Server Agents. Die einzelnen Publikationen können dabei flexibel entweder zusammen in einem Job oder aber zu unterschiedlichen Intervallen über individuelle Jobs bearbeitet werden. Der Aufruf sieht bei einem gemeinsamen Job so aus:

```
use RZNW_LocImport1
exec dbo.spMsApplyMainExportLogSnaphot
exec dbo.spMsApplyMainExportRecipeSnaphot
exec dbo.spMsApplyMainExportMachAdminSnaphot
```

7 Zusammenfassung und Ausblick

Das Ziel dieser Arbeit war die konkrete Ausarbeitung einer Datenbanksynchronisationslösung, in der der Benutzer volle Kontrolle über die Übernahme oder Ablehnung aller Änderungen von einer Standort-Datenbank in eine zentrale Datenbank hat.

Zuerst wurde die Datenbanken Thematik durch grundlegende Begriffsbestimmungen erläutert. Es wurde aufgezeigt mit welchen Modellen und Methoden reale Objekte in einer Datenbank abgebildet werden können. Dies war deshalb wichtig, damit ausgehend vom RZNW ERD die Zusammenhänge innerhalb einer RZNW Datenbank interpretiert werden können.

Ein wesentliches Kapitel befasste sich mit der Replikation und welche grundsätzlichen Replikationsverfahren in der Literatur für die Synchronisation von Datenbanken diskutiert werden.

Der zweite Teil der Arbeit behandelte die Ausarbeitung einer Datenbanksynchronisationslösung auf Grundlage von konkreten Kundenanforderungen.

Hier war eine zentrale designentscheidende Kundenforderung, Änderungen von den Standortdatenbanken **vor** der definitiven Übernahme in die zentrale Datenbank zu kontrollieren. Aus diesem Grunde konnte nicht ein einfaches Replikationsszenario mit Verleger, Publikationen, Abonnenten und Abonnement eingesetzt werden, sondern es bedarf eines ausgefeilten, mehrstufigen Konzeptes.

Das Problem dieser Anforderungen war, dass nur für einen kleinen Teil auf die von den MS SQL Server Versionen unterstützten Replikationsmechanismen und Funktionen zurückgegriffen werden konnte.

Die Replikationslösung RZNW-Multiserver ist beim Kunden TKT-Presta seit gut vier Monaten in Betrieb und seit drei Monaten auch voll funktionsfähig.

Die Lösung an sich ist trotz ihrer doch beachtlichen Komplexität noch relativ transparent.

In der Praxis konnten schon die ersten Designverbesserungsvorschläge ausgiebig erprobt werden. Die vorgestellte Lösung funktioniert für Anwender, die den grundsätzlichen Aufbau der RZNW Datenbank kennen, recht gut. Werden die Datenbanken aber nicht zeitnah synchronisiert dann kommt es zu Konflikten bei der Anwendung der Schnappschüsse auf die RZNW Datenbanken die nur über SQL gelöst werden können. Bisher wurden Probleme mit der Merge-Replikation durch doppelt auftretende Merge-Indexe korrigiert. Die Einstellungen der beteiligten Replikations-Agenten sind noch nicht optimiert. Ein erstes brauchbares Gleichgewicht aus dem Zielkonflikt der Replikation zwischen Verfügbarkeit, der Summe der Änderungsoperationen und der Datenkonsistenz ist im damit gefunden, kann aber erst durch die Praxis weiter verfeinert werden.

Anlagen

Anlage 1 ist das RZNW ERD.

Anlage 2 sind die 12 goldenen Regeln für relationale Datenbanken von Dr. E. F. Codd

Anlage 3 ist als Beispiel die Erstellung Skripte der Snapshot Publikationen am STAO

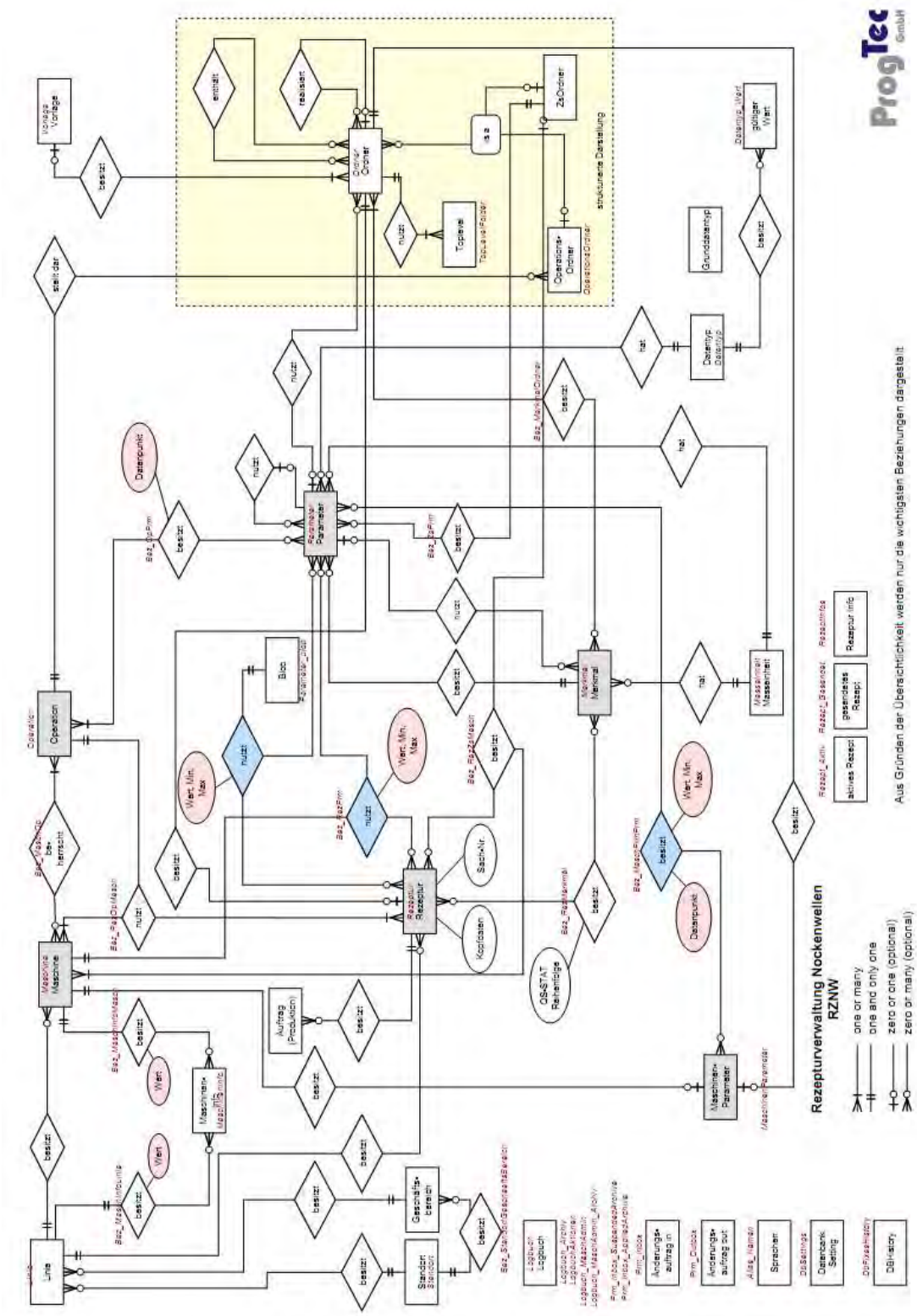
Anlage 4 Beispiel zur Schnappschuss Erstellung des MachAdmin in die Export DB aus der Standort Datenbank

Anlage 5 Beispiel des Anwendung des Rezeptur Schnappschusses vom STAO auf die HS Datenbank.

ACHTUNG: Die kompletten Projektsourcen sind nur in elektronischer Form zu dieser Diplomarbeit verfügbar.

Anlage 1

RZNW-ERD



Anlage 2

„Die 12 goldenen Regeln für relationale Datenbanken[96]

Das bekannteste Modell zur Datenspeicherung ist die relationale Datenbank, die auf den Artikel **»A Relational Model of Data for Large Shared Data Banks « von Dr. E. F. Codd** aus dem Jahr 1970 zurückgeht. Die Entwicklung von SQL zielt darauf ab, den Konzepten des relationalen Datenbankmodells zu entsprechen. Dr. Codd veröffentlichte 12 Regeln (inklusive Regel 0 eigentlich 13), die eine relationale Datenbank im strengen Sinne definieren:

Ein relationales DBMS muß in der Lage sein, Datenbanken vollständig über seine relationalen Fähigkeiten zu verwalten.

Regel 1: Darstellung von Informationen

Alle Informationen in einer relationalen Datenbank (einschließlich Namen von Tabellen und Spalten) sind explizit als Werte in Tabellen darzustellen.

Regel 2: Zugriff auf Daten

Jeder Wert einer relationalen Datenbank muß durch eine Kombination von Tabellenname, Primärschlüssel und Spaltenname auffindbar sein.

Regel 3: Systematische Behandlung von Nullwerten

Das DBMS behandelt Nullwerte durchgängig gleich als unbekannte oder fehlende Daten und unterscheidet diese von Standardwerten.

Regel 4: Struktur einer Datenbank

Die Datenbank und ihre Inhalte werden in einem sogenannten Systemkatalog auf derselben logischen Ebene wie die Daten selbst - also in Tabellen - beschrieben. Demzufolge läßt sich der Katalog mit Hilfe der Datenbanksprache abfragen.

Regel 5: Abfragesprache

Zu einem relationalen System gehört mindestens eine Abfragesprache mit einem vollständigen Befehlssatz für Datendefinition, Manipulation, Integritätsregeln, Autorisierung und Transaktionen.

---

[96] /Codd 85/

Regel 6: Aktualisieren von Sichten

Alle Sichten, die theoretisch aktualisiert werden können, lassen sich auch vom System aktualisieren.

Regel 7: Abfragen und Bearbeiten ganzer Tabellen

Das DBMS unterstützt nicht nur Abfragen, sondern auch die Operationen für Einfügen, Aktualisieren und Löschen in Form ganzer Tabellen.

Regel 8: Physikalische Datenunabhängigkeit

Der logische Zugriff auf die Daten durch Anwendungen und Ad-Hoc-Programme muß unabhängig von den physikalischen Zugriffsmethoden oder den Speicherstrukturen der Daten sein.

Regel 9: Logische Datenunabhängigkeit

Änderungen der Tabellenstrukturen dürfen keinen Einfluß auf die Logik der Anwendungen und Ad-Hoc-Programme haben.

Regel 10: Unabhängigkeit der Integrität

Integritätsregeln müssen sich in der Datenbanksprache definieren lassen. Die Regeln müssen im Systemkatalog gespeichert werden. Es darf nicht möglich sein, die Regeln zu umgehen.

Regel 11: Verteilungsunabhängigkeit

Der logische Zugriff auf die Daten durch Anwendungen und Ad-Hoc-Programme darf sich beim Übergang von einer nicht-verteilten zu einer verteilten Datenbank nicht ändern.

Regel 12: Kein Unterlaufen der Abfragesprache

Integritätsregeln, die über die Datenbanksprache definiert sind, dürfen sich nicht mit Hilfe von Low-Level-Sprachen umgehen lassen.

Anlage 3

Snapshot Publikationen am STAO

```
-- Enabling the replication database
use master
GO

exec sp_replicationdboption @dbname = N'RZNW-LocExport', @optname =
N'publish', @value = N'true'
GO

use [RZNW-LocExport]
GO

-- Adding the snapshot publication
exec sp_addpublication @publication = N'Location-Logs', @restricted =
N'false', @sync_method = N'native', @repl_freq = N'snapshot', @description
= N'Snapshot publication of RZNW Loacation Server Database''s Own Logs',
@status = N'active', @allow_push = N'true', @allow_pull = N'true',
@allow_anonymous = N'false', @enabled_for_internet = N'false',
@independent_agent = N'false', @immediate_sync = N'false', @allow_sync_tran
= N'false', @autogen_sync_procs = N'false', @retention = 336,
@allow_queued_tran = N'false', @snapshot_in_defaultfolder = N'true',
@compress_snapshot = N'false', @ftp_port = 21, @ftp_login = N'anonymous',
@allow_dts = N'false', @allow_subscription_copy = N'false',
@add_to_active_directory = N'false'
exec sp_addpublication_snapshot @publication = N'Location-
Logs',@frequency_type = 4, @frequency_interval = 1,
@frequency_relative_interval = 1, @frequency_recurrence_factor = 0,
@frequency_subday = 8, @frequency_subday_interval = 1, @active_start_date =
0, @active_end_date = 0, @active_start_time_of_day = 0,
@active_end_time_of_day = 235959, @snapshot_job_name = N'VM-2K-RZNW-
LocExport-Location-Logs-3'
GO

exec sp_grant_publication_access @publication = N'Location-Logs', @login =
N'distributor_admin'
GO
exec sp_grant_publication_access @publication = N'Location-Logs', @login =
N'sa'
GO
exec sp_grant_publication_access @publication = N'Location-Logs', @login =
N'builtin\administrators'
GO

-- Adding the snapshot articles
exec sp_addarticle @publication = N'Location-Logs', @article = N'Logbuch',
@source_owner = N'dbo', @source_object = N'Logbuch', @destination_table =
N'Logbuch', @type = N'logbased', @creation_script = null, @description =
null, @pre_creation_cmd = N'drop', @schema_option = 0x00000000000000F1,
@status = 0, @vertical_partition = N'false', @ins_cmd = N'SQL', @del_cmd =
N'SQL', @upd_cmd = N'SQL', @filter = null, @sync_object = null,
@auto_identity_range = N'false'
GO
exec sp_addarticle @publication = N'Location-Logs', @article =
N'Logbuch_MaschAdmin', @source_owner = N'dbo', @source_object =
N'Logbuch_MaschAdmin', @destination_table = N'Logbuch_MaschAdmin', @type =
N'logbased', @creation_script = null, @description = null,
```

```
@pre_creation_cmd = N'drop', @schema_option = 0x00000000000000F1, @status =
0, @vertical_partition = N'false', @ins_cmd = N'SQL', @del_cmd = N'SQL',
@upd_cmd = N'SQL', @filter = null, @sync_object = null,
@auto_identity_range = N'false'
GO
exec sp_addarticle @publication = N'Location-Logs', @article =
N'SnapshotInfo', @source_owner = N'dbo', @source_object = N'SnapshotInfo',
@destination_table = N'SnapshotInfo', @type = N'logbased', @creation_script
= null, @description = null, @pre_creation_cmd = N'drop', @schema_option =
0x00000000000000F1, @status = 0, @vertical_partition = N'false', @ins_cmd =
N'SQL', @del_cmd = N'SQL', @upd_cmd = N'SQL', @filter = null, @sync_object
= null, @auto_identity_range = N'false'
GO
exec sp_addarticle @publication = N'Location-Logs', @article =
N'AutoLog_Templates', @source_owner = N'dbo', @source_object =
N'AutoLog_Templates', @destination_table = N'AutoLog_Templates', @type =
N'logbased', @creation_script = null, @description = null,
@pre_creation_cmd = N'drop', @schema_option = 0x00000000000000F1, @status =
0, @vertical_partition = N'false', @ins_cmd = N'SQL', @del_cmd = N'SQL',
@upd_cmd = N'SQL', @filter = null, @sync_object = null,
@auto_identity_range = N'false'
GO
exec sp_addarticle @publication = N'Location-Logs', @article = N'DbInfo',
@source_owner = N'dbo', @source_object = N'DbInfo', @destination_table =
N'DbInfo', @type = N'logbased', @creation_script = null, @description =
null, @pre_creation_cmd = N'drop', @schema_option = 0x00000000000000F1,
@status = 0, @vertical_partition = N'false', @ins_cmd = N'SQL', @del_cmd =
N'SQL', @upd_cmd = N'SQL', @filter = null, @sync_object = null,
@auto_identity_range = N'false'
GO


use [RZNW-LocExport]
GO

-- Adding the snapshot publication
exec sp_addpublication @publication = N'Location-MachineAdmin', @restricted
= N'false', @sync_method = N'native', @repl_freq = N'snapshot',
@description = N'Snapshot publication of RZNW Loacation Server Database''s
Own Machine Administration', @status = N'active', @allow_push = N'true',
@allow_pull = N'true', @allow_anonymous = N'false', @enabled_for_internet =
N'false', @independent_agent = N'false', @immediate_sync = N'false',
@allow_sync_tran = N'false', @autogen_sync_procs = N'false', @retention =
336, @allow_queued_tran = N'false', @snapshot_in_defaultfolder = N'true',
@compress_snapshot = N'false', @ftp_port = 21, @ftp_login = N'anonymous',
@allow_dts = N'false', @allow_subscription_copy = N'false',
@add_to_active_directory = N'false'
exec sp_addpublication_snapshot @publication = N'Location-
MachineAdmin',@frequency_type = 4, @frequency_interval = 1,
@frequency_relative_interval = 1, @frequency_recurrence_factor = 0,
@frequency_subday = 8, @frequency_subday_interval = 1, @active_start_date =
0, @active_end_date = 0, @active_start_time_of_day = 0,
@active_end_time_of_day = 235959, @snapshot_job_name = N'VM-2K-RZNW-
LocExport-Location-MachineAdmin-4'
GO

exec sp_grant_publication_access @publication = N'Location-MachineAdmin',
@login = N'distributor_admin'
GO
exec sp_grant_publication_access @publication = N'Location-MachineAdmin',
@login = N'sa'
GO
```

```
exec sp_grant_publication_access @publication = N'Location-MachineAdmin',
@login = N'builtin\administrators'
GO

-- Adding the snapshot articles
exec sp_addarticle @publication = N'Location-MachineAdmin', @article =
N'Alias_Namen_MachAdmin', @source_owner = N'dbo', @source_object =
N'Alias_Namen_MachAdmin', @destination_table = N'Alias_Namen_MachAdmin',
@type = N'logbased', @creation_script = null, @description = null,
@pre_creation_cmd = N'drop', @schema_option = 0x00000000000000F1, @status =
0, @vertical_partition = N'false', @ins_cmd = N'SQL', @del_cmd = N'SQL',
@upd_cmd = N'SQL', @filter = null, @sync_object = null,
@auto_identity_range = N'false'
GO
exec sp_addarticle @publication = N'Location-MachineAdmin', @article =
N'Bez_MaschInfoLinie', @source_owner = N'dbo', @source_object =
N'Bez_MaschInfoLinie', @destination_table = N'Bez_MaschInfoLinie', @type =
N'logbased', @creation_script = null, @description = null,
@pre_creation_cmd = N'drop', @schema_option = 0x00000000000000F1, @status =
0, @vertical_partition = N'false', @ins_cmd = N'SQL', @del_cmd = N'SQL',
@upd_cmd = N'SQL', @filter = null, @sync_object = null,
@auto_identity_range = N'false'
GO
exec sp_addarticle @publication = N'Location-MachineAdmin', @article =
N'Bez_MaschInfoMasch', @source_owner = N'dbo', @source_object =
N'Bez_MaschInfoMasch', @destination_table = N'Bez_MaschInfoMasch', @type =
N'logbased', @creation_script = null, @description = null,
@pre_creation_cmd = N'drop', @schema_option = 0x00000000000000F1, @status =
0, @vertical_partition = N'false', @ins_cmd = N'SQL', @del_cmd = N'SQL',
@upd_cmd = N'SQL', @filter = null, @sync_object = null,
@auto_identity_range = N'false'
GO
exec sp_addarticle @publication = N'Location-MachineAdmin', @article =
N'Bez_MaschOp', @source_owner = N'dbo', @source_object = N'Bez_MaschOp',
@destination_table = N'Bez_MaschOp', @type = N'logbased', @creation_script
= null, @description = null, @pre_creation_cmd = N'drop', @schema_option =
0x00000000000000F1, @status = 0, @vertical_partition = N'false', @ins_cmd =
N'SQL', @del_cmd = N'SQL', @upd_cmd = N'SQL', @filter = null, @sync_object
= null, @auto_identity_range = N'false'
GO
exec sp_addarticle @publication = N'Location-MachineAdmin', @article =
N'Linie', @source_owner = N'dbo', @source_object = N'Linie',
@destination_table = N'Linie', @type = N'logbased', @creation_script =
null, @description = null, @pre_creation_cmd = N'drop', @schema_option =
0x00000000000000F1, @status = 0, @vertical_partition = N'false', @ins_cmd =
N'SQL', @del_cmd = N'SQL', @upd_cmd = N'SQL', @filter = null, @sync_object
= null, @auto_identity_range = N'false'
GO
exec sp_addarticle @publication = N'Location-MachineAdmin', @article =
N'LocalRemovedElements', @source_owner = N'dbo', @source_object =
N'LocalRemovedElements', @destination_table = N'LocalRemovedElements',
@type = N'logbased', @creation_script = null, @description = null,
@pre_creation_cmd = N'drop', @schema_option = 0x00000000000000F1, @status =
0, @vertical_partition = N'false', @ins_cmd = N'SQL', @del_cmd = N'SQL',
@upd_cmd = N'SQL', @filter = null, @sync_object = null,
@auto_identity_range = N'false'
GO
exec sp_addarticle @publication = N'Location-MachineAdmin', @article =
N'Maschine', @source_owner = N'dbo', @source_object = N'Maschine',
@destination_table = N'Maschine', @type = N'logbased', @creation_script =
null, @description = null, @pre_creation_cmd = N'drop', @schema_option =
0x00000000000000F1, @status = 0, @vertical_partition = N'false', @ins_cmd =
```

```
N'SQL', @del_cmd = N'SQL', @upd_cmd = N'SQL', @filter = null, @sync_object
= null, @auto_identity_range = N'false'
GO
exec sp_addarticle @publication = N'Location-MachineAdmin', @article =
N'MaschinenInfo', @source_owner = N'dbo', @source_object =
N'MaschinenInfo', @destination_table = N'MaschinenInfo', @type =
N'logbased', @creation_script = null, @description = null,
@pre_creation_cmd = N'drop', @schema_option = 0x00000000000000F1, @status =
0, @vertical_partition = N'false', @ins_cmd = N'SQL', @del_cmd = N'SQL',
@upd_cmd = N'SQL', @filter = null, @sync_object = null,
@auto_identity_range = N'false'
GO
exec sp_addarticle @publication = N'Location-MachineAdmin', @article =
N'MaschinenInfoWert', @source_owner = N'dbo', @source_object =
N'MaschinenInfoWert', @destination_table = N'MaschinenInfoWert', @type =
N'logbased', @creation_script = null, @description = null,
@pre_creation_cmd = N'drop', @schema_option = 0x00000000000000F1, @status =
0, @vertical_partition = N'false', @ins_cmd = N'SQL', @del_cmd = N'SQL',
@upd_cmd = N'SQL', @filter = null, @sync_object = null,
@auto_identity_range = N'false'
GO
exec sp_addarticle @publication = N'Location-MachineAdmin', @article =
N'Ordner_MachAdmin', @source_owner = N'dbo', @source_object =
N'Ordner_MachAdmin', @destination_table = N'Ordner_MachAdmin', @type =
N'logbased', @creation_script = null, @description = null,
@pre_creation_cmd = N'drop', @schema_option = 0x00000000000000F1, @status =
0, @vertical_partition = N'false', @ins_cmd = N'SQL', @del_cmd = N'SQL',
@upd_cmd = N'SQL', @filter = null, @sync_object = null,
@auto_identity_range = N'false'
GO
exec sp_addarticle @publication = N'Location-MachineAdmin', @article =
N'SnapshotInfo', @source_owner = N'dbo', @source_object = N'SnapshotInfo',
@destination_table = N'SnapshotInfo', @type = N'logbased', @creation_script
= null, @description = null, @pre_creation_cmd = N'drop', @schema_option =
0x00000000000000F1, @status = 0, @vertical_partition = N'false', @ins_cmd =
N'SQL', @del_cmd = N'SQL', @upd_cmd = N'SQL', @filter = null, @sync_object
= null, @auto_identity_range = N'false'
GO


use [RZNW-LocExport]
GO

-- Adding the snapshot publication
exec sp_addpublication @publication = N'Location-Recipes', @restricted =
N'false', @sync_method = N'native', @repl_freq = N'snapshot', @description
= N'Snapshot publication of RZNW Loacation Server Database''s Own Recipe
Modifications', @status = N'active', @allow_push = N'true', @allow_pull =
N'true', @allow_anonymous = N'false', @enabled_for_internet = N'false',
@independent_agent = N'false', @immediate_sync = N'false', @allow_sync_tran
= N'false', @autogen_sync_procs = N'false', @retention = 336,
@allow_queued_tran = N'false', @snapshot_in_defaultfolder = N'true',
@compress_snapshot = N'false', @ftp_port = 21, @ftp_login = N'anonymous',
@allow_dts = N'false', @allow_subscription_copy = N'false',
@add_to_active_directory = N'false'
exec sp_addpublication_snapshot @publication = N'Location-
Recipes',@frequency_type = 4, @frequency_interval = 1,
@frequency_relative_interval = 1, @frequency_recurrence_factor = 0,
@frequency_subday = 8, @frequency_subday_interval = 1, @active_start_date =
0, @active_end_date = 0, @active_start_time_of_day = 0,
@active_end_time_of_day = 235959, @snapshot_job_name = N'VM-2K-RZNW-
LocExport-Location-Recipes-2'
```

```
GO

exec sp_grant_publication_access @publication = N'Location-Recipes', @login
= N'distributor_admin'
GO
exec sp_grant_publication_access @publication = N'Location-Recipes', @login
= N'sa'
GO
exec sp_grant_publication_access @publication = N'Location-Recipes', @login
= N'builtin\administrators'
GO

-- Adding the snapshot articles
exec sp_addarticle @publication = N'Location-Recipes', @article =
N'Bez_MaschPrmPrm', @source_owner = N'dbo', @source_object =
N'Bez_MaschPrmPrm', @destination_table = N'Bez_MaschPrmPrm', @type =
N'logbased', @creation_script = null, @description = null,
@pre_creation_cmd = N'drop', @schema_option = 0x00000000000000F1, @status =
0, @vertical_partition = N'false', @ins_cmd = N'SQL', @del_cmd = N'SQL',
@upd_cmd = N'SQL', @filter = null, @sync_object = null,
@auto_identity_range = N'false'
GO
exec sp_addarticle @publication = N'Location-Recipes', @article =
N'Bez_MerkmalOrdner_Rcp', @source_owner = N'dbo', @source_object =
N'Bez_MerkmalOrdner_Rcp', @destination_table = N'Bez_MerkmalOrdner_Rcp',
@type = N'logbased', @creation_script = null, @description = null,
@pre_creation_cmd = N'drop', @schema_option = 0x00000000000000F1, @status =
0, @vertical_partition = N'false', @ins_cmd = N'SQL', @del_cmd = N'SQL',
@upd_cmd = N'SQL', @filter = null, @sync_object = null,
@auto_identity_range = N'false'
GO
exec sp_addarticle @publication = N'Location-Recipes', @article =
N'Bez_RezMerkmal', @source_owner = N'dbo', @source_object =
N'Bez_RezMerkmal', @destination_table = N'Bez_RezMerkmal', @type =
N'logbased', @creation_script = null, @description = null,
@pre_creation_cmd = N'drop', @schema_option = 0x00000000000000F1, @status =
0, @vertical_partition = N'false', @ins_cmd = N'SQL', @del_cmd = N'SQL',
@upd_cmd = N'SQL', @filter = null, @sync_object = null,
@auto_identity_range = N'false'
GO
exec sp_addarticle @publication = N'Location-Recipes', @article =
N'Bez_RezOpMasch', @source_owner = N'dbo', @source_object =
N'Bez_RezOpMasch', @destination_table = N'Bez_RezOpMasch', @type =
N'logbased', @creation_script = null, @description = null,
@pre_creation_cmd = N'drop', @schema_option = 0x00000000000000F1, @status =
0, @vertical_partition = N'false', @ins_cmd = N'SQL', @del_cmd = N'SQL',
@upd_cmd = N'SQL', @filter = null, @sync_object = null,
@auto_identity_range = N'false'
GO
exec sp_addarticle @publication = N'Location-Recipes', @article =
N'Bez_RezPrm', @source_owner = N'dbo', @source_object = N'Bez_RezPrm',
@destination_table = N'Bez_RezPrm', @type = N'logbased', @creation_script =
null, @description = null, @pre_creation_cmd = N'drop', @schema_option =
0x00000000000000F1, @status = 0, @vertical_partition = N'false', @ins_cmd =
N'SQL', @del_cmd = N'SQL', @upd_cmd = N'SQL', @filter = null, @sync_object
= null, @auto_identity_range = N'false'
GO
exec sp_addarticle @publication = N'Location-Recipes', @article =
N'Bez_RezZsMasch', @source_owner = N'dbo', @source_object =
N'Bez_RezZsMasch', @destination_table = N'Bez_RezZsMasch', @type =
N'logbased', @creation_script = null, @description = null,
@pre_creation_cmd = N'drop', @schema_option = 0x00000000000000F1, @status =
```

```
0, @vertical_partition = N'false', @ins_cmd = N'SQL', @del_cmd = N'SQL',
@upd_cmd = N'SQL', @filter = null, @sync_object = null,
@auto_identity_range = N'false'
GO
exec sp_addarticle @publication = N'Location-Recipes', @article =
N'LocalRemovedElements', @source_owner = N'dbo', @source_object =
N'LocalRemovedElements', @destination_table = N'LocalRemovedElements',
@type = N'logbased', @creation_script = null, @description = null,
@pre_creation_cmd = N'drop', @schema_option = 0x00000000000000F1, @status =
0, @vertical_partition = N'false', @ins_cmd = N'SQL', @del_cmd = N'SQL',
@upd_cmd = N'SQL', @filter = null, @sync_object = null,
@auto_identity_range = N'false'
GO
exec sp_addarticle @publication = N'Location-Recipes', @article =
N'MaschinenParameter', @source_owner = N'dbo', @source_object =
N'MaschinenParameter', @destination_table = N'MaschinenParameter', @type =
N'logbased', @creation_script = null, @description = null,
@pre_creation_cmd = N'drop', @schema_option = 0x00000000000000F1, @status =
0, @vertical_partition = N'false', @ins_cmd = N'SQL', @del_cmd = N'SQL',
@upd_cmd = N'SQL', @filter = null, @sync_object = null,
@auto_identity_range = N'false'
GO
exec sp_addarticle @publication = N'Location-Recipes', @article =
N'Ordner_Rcp', @source_owner = N'dbo', @source_object = N'Ordner_Rcp',
@destination_table = N'Ordner_Rcp', @type = N'logbased', @creation_script =
null, @description = null, @pre_creation_cmd = N'drop', @schema_option =
0x00000000000000F1, @status = 0, @vertical_partition = N'false', @ins_cmd =
N'SQL', @del_cmd = N'SQL', @upd_cmd = N'SQL', @filter = null, @sync_object
= null, @auto_identity_range = N'false'
GO
exec sp_addarticle @publication = N'Location-Recipes', @article =
N'Parameter_Blob', @source_owner = N'dbo', @source_object =
N'Parameter_Blob', @destination_table = N'Parameter_Blob', @type =
N'logbased', @creation_script = null, @description = null,
@pre_creation_cmd = N'drop', @schema_option = 0x00000000000000F1, @status =
0, @vertical_partition = N'false', @ins_cmd = N'SQL', @del_cmd = N'SQL',
@upd_cmd = N'SQL', @filter = null, @sync_object = null,
@auto_identity_range = N'false'
GO
exec sp_addarticle @publication = N'Location-Recipes', @article =
N'Rezeptur', @source_owner = N'dbo', @source_object = N'Rezeptur',
@destination_table = N'Rezeptur', @type = N'logbased', @creation_script =
null, @description = null, @pre_creation_cmd = N'drop', @schema_option =
0x00000000000000F1, @status = 0, @vertical_partition = N'false', @ins_cmd =
N'SQL', @del_cmd = N'SQL', @upd_cmd = N'SQL', @filter = null, @sync_object
= null, @auto_identity_range = N'false'
GO
exec sp_addarticle @publication = N'Location-Recipes', @article =
N'SnapshotInfo', @source_owner = N'dbo', @source_object = N'SnapshotInfo',
@destination_table = N'SnapshotInfo', @type = N'logbased', @creation_script
= null, @description = null, @pre_creation_cmd = N'drop', @schema_option =
0x00000000000000F1, @status = 0, @vertical_partition = N'false', @ins_cmd =
N'SQL', @del_cmd = N'SQL', @upd_cmd = N'SQL', @filter = null, @sync_object
= null, @auto_identity_range = N'false'
GO


use [RZNW-LocExport]
GO

-- Adding the snapshot publication
```

```
exec sp_addpublication @publication = N'Location-Templates', @restricted =
N'false', @sync_method = N'native', @repl_freq = N'snapshot', @description
= N'Snapshot publication of RZNW Loacation Server Database''s Template
Modifications', @status = N'active', @allow_push = N'true', @allow_pull =
N'true', @allow_anonymous = N'false', @enabled_for_internet = N'false',
@independent_agent = N'false', @immediate_sync = N'false', @allow_sync_tran
= N'false', @autogen_sync_procs = N'false', @retention = 336,
@allow_queued_tran = N'false', @snapshot_in_defaultfolder = N'true',
@compress_snapshot = N'false', @ftp_port = 21, @ftp_login = N'anonymous',
@allow_dts = N'false', @allow_subscription_copy = N'false',
@add_to_active_directory = N'false'
exec sp_addpublication_snapshot @publication = N'Location-
Templates',@frequency_type = 4, @frequency_interval = 1,
@frequency_relative_interval = 0, @frequency_recurrence_factor = 1,
@frequency_subday = 1, @frequency_subday_interval = 0, @active_start_date =
0, @active_end_date = 0, @active_start_time_of_day = 232200,
@active_end_time_of_day = 0, @snapshot_job_name = N'VM-2K-RZNW-LocExport-
Location-Templates-1'
GO

exec sp_grant_publication_access @publication = N'Location-Templates',
@login = N'distributor_admin'
GO
exec sp_grant_publication_access @publication = N'Location-Templates',
@login = N'sa'
GO
exec sp_grant_publication_access @publication = N'Location-Templates',
@login = N'builtin\administrators'
GO

-- Adding the snapshot articles
exec sp_addarticle @publication = N'Location-Templates', @article =
N'Alias_Namen', @source_owner = N'dbo', @source_object = N'Alias_Namen',
@destination_table = N'Alias_Namen', @type = N'logbased', @creation_script
= null, @description = null, @pre_creation_cmd = N'drop', @schema_option =
0x00000000000000F1, @status = 0, @vertical_partition = N'false', @ins_cmd =
N'SQL', @del_cmd = N'SQL', @upd_cmd = N'SQL', @filter = null, @sync_object
= null, @auto_identity_range = N'false'
GO
exec sp_addarticle @publication = N'Location-Templates', @article =
N'Bez_MaschOp', @source_owner = N'dbo', @source_object = N'Bez_MaschOp',
@destination_table = N'Bez_MaschOp', @type = N'logbased', @creation_script
= null, @description = null, @pre_creation_cmd = N'drop', @schema_option =
0x00000000000000F1, @status = 0, @vertical_partition = N'false', @ins_cmd =
N'SQL', @del_cmd = N'SQL', @upd_cmd = N'SQL', @filter = null, @sync_object
= null, @auto_identity_range = N'false'
GO
exec sp_addarticle @publication = N'Location-Templates', @article =
N'Bez_MerkmalOrdner', @source_owner = N'dbo', @source_object =
N'Bez_MerkmalOrdner', @destination_table = N'Bez_MerkmalOrdner', @type =
N'logbased', @creation_script = null, @description = null,
@pre_creation_cmd = N'drop', @schema_option = 0x00000000000000F1, @status =
0, @vertical_partition = N'false', @ins_cmd = N'SQL', @del_cmd = N'SQL',
@upd_cmd = N'SQL', @filter = null, @sync_object = null,
@auto_identity_range = N'false'
GO
exec sp_addarticle @publication = N'Location-Templates', @article =
N'Bez_OpPrm', @source_owner = N'dbo', @source_object = N'Bez_OpPrm',
@destination_table = N'Bez_OpPrm', @type = N'logbased', @creation_script =
null, @description = null, @pre_creation_cmd = N'drop', @schema_option =
0x00000000000000F1, @status = 0, @vertical_partition = N'false', @ins_cmd =
```

```
N'SQL', @del_cmd = N'SQL', @upd_cmd = N'SQL', @filter = null, @sync_object
= null, @auto_identity_range = N'false'
GO
exec sp_addarticle @publication = N'Location-Templates', @article =
N'Bez_ZsPrm', @source_owner = N'dbo', @source_object = N'Bez_ZsPrm',
@destination_table = N'Bez_ZsPrm', @type = N'logbased', @creation_script =
null, @description = null, @pre_creation_cmd = N'drop', @schema_option =
0x00000000000000F1, @status = 0, @vertical_partition = N'false', @ins_cmd =
N'SQL', @del_cmd = N'SQL', @upd_cmd = N'SQL', @filter = null, @sync_object
= null, @auto_identity_range = N'false'
GO
exec sp_addarticle @publication = N'Location-Templates', @article =
N'Datentyp', @source_owner = N'dbo', @source_object = N'Datentyp',
@destination_table = N'Datentyp', @type = N'logbased', @creation_script =
null, @description = null, @pre_creation_cmd = N'drop', @schema_option =
0x00000000000000F1, @status = 0, @vertical_partition = N'false', @ins_cmd =
N'SQL', @del_cmd = N'SQL', @upd_cmd = N'SQL', @filter = null, @sync_object
= null, @auto_identity_range = N'false'
GO
exec sp_addarticle @publication = N'Location-Templates', @article =
N'Datentyp_Wert', @source_owner = N'dbo', @source_object =
N'Datentyp_Wert', @destination_table = N'Datentyp_Wert', @type =
N'logbased', @creation_script = null, @description = null,
@pre_creation_cmd = N'drop', @schema_option = 0x00000000000000F1, @status =
0, @vertical_partition = N'false', @ins_cmd = N'SQL', @del_cmd = N'SQL',
@upd_cmd = N'SQL', @filter = null, @sync_object = null,
@auto_identity_range = N'false'
GO
exec sp_addarticle @publication = N'Location-Templates', @article =
N'Masseinheit', @source_owner = N'dbo', @source_object = N'Masseinheit',
@destination_table = N'Masseinheit', @type = N'logbased', @creation_script
= null, @description = null, @pre_creation_cmd = N'drop', @schema_option =
0x00000000000000F1, @status = 0, @vertical_partition = N'false', @ins_cmd =
N'SQL', @del_cmd = N'SQL', @upd_cmd = N'SQL', @filter = null, @sync_object
= null, @auto_identity_range = N'false'
GO
exec sp_addarticle @publication = N'Location-Templates', @article =
N'Merkmal', @source_owner = N'dbo', @source_object = N'Merkmal',
@destination_table = N'Merkmal', @type = N'logbased', @creation_script =
null, @description = null, @pre_creation_cmd = N'drop', @schema_option =
0x00000000000000F1, @status = 0, @vertical_partition = N'false', @ins_cmd =
N'SQL', @del_cmd = N'SQL', @upd_cmd = N'SQL', @filter = null, @sync_object
= null, @auto_identity_range = N'false'
GO
exec sp_addarticle @publication = N'Location-Templates', @article =
N'Operation', @source_owner = N'dbo', @source_object = N'Operation',
@destination_table = N'Operation', @type = N'logbased', @creation_script =
null, @description = null, @pre_creation_cmd = N'drop', @schema_option =
0x00000000000000F1, @status = 0, @vertical_partition = N'false', @ins_cmd =
N'SQL', @del_cmd = N'SQL', @upd_cmd = N'SQL', @filter = null, @sync_object
= null, @auto_identity_range = N'false'
GO
exec sp_addarticle @publication = N'Location-Templates', @article =
N'OperationsOrdner', @source_owner = N'dbo', @source_object =
N'OperationsOrdner', @destination_table = N'OperationsOrdner', @type =
N'logbased', @creation_script = null, @description = null,
@pre_creation_cmd = N'drop', @schema_option = 0x00000000000000F1, @status =
0, @vertical_partition = N'false', @ins_cmd = N'SQL', @del_cmd = N'SQL',
@upd_cmd = N'SQL', @filter = null, @sync_object = null,
@auto_identity_range = N'false'
GO
```

```
exec sp_addarticle @publication = N'Location-Templates', @article =
N'Ordner', @source_owner = N'dbo', @source_object = N'Ordner',
@destination_table = N'Ordner', @type = N'logbased', @creation_script =
null, @description = null, @pre_creation_cmd = N'drop', @schema_option =
0x00000000000000F1, @status = 0, @vertical_partition = N'false', @ins_cmd =
N'SQL', @del_cmd = N'SQL', @upd_cmd = N'SQL', @filter = null, @sync_object
= null, @auto_identity_range = N'false'
GO
exec sp_addarticle @publication = N'Location-Templates', @article =
N'Parameter', @source_owner = N'dbo', @source_object = N'Parameter',
@destination_table = N'Parameter', @type = N'logbased', @creation_script =
null, @description = null, @pre_creation_cmd = N'drop', @schema_option =
0x00000000000000F1, @status = 0, @vertical_partition = N'false', @ins_cmd =
N'SQL', @del_cmd = N'SQL', @upd_cmd = N'SQL', @filter = null, @sync_object
= null, @auto_identity_range = N'false'
GO
exec sp_addarticle @publication = N'Location-Templates', @article =
N'SnapshotInfo', @source_owner = N'dbo', @source_object = N'SnapshotInfo',
@destination_table = N'SnapshotInfo', @type = N'logbased', @creation_script
= null, @description = null, @pre_creation_cmd = N'drop', @schema_option =
0x00000000000000F1, @status = 0, @vertical_partition = N'false', @ins_cmd =
N'SQL', @del_cmd = N'SQL', @upd_cmd = N'SQL', @filter = null, @sync_object
= null, @auto_identity_range = N'false'
GO
exec sp_addarticle @publication = N'Location-Templates', @article =
N'ZsOrdner', @source_owner = N'dbo', @source_object = N'ZsOrdner',
@destination_table = N'ZsOrdner', @type = N'logbased', @creation_script =
null, @description = null, @pre_creation_cmd = N'drop', @schema_option =
0x00000000000000F1, @status = 0, @vertical_partition = N'false', @ins_cmd =
N'SQL', @del_cmd = N'SQL', @upd_cmd = N'SQL', @filter = null, @sync_object
= null, @auto_identity_range = N'false'
GO


-- Enabling the replication database
use master
GO

exec sp_replicationdboption @dbname = N'RZNW-LocExport', @optname = N'merge
publish', @value = N'true'
GO

use [RZNW-LocExport]
GO

-- Adding the merge publication
exec sp_addmergepublication @publication = N'Sync_Log', @description =
N'Merge publication of RZNW Location Server Database''s Synchronization
Information with the Main Server', @retention = 14, @sync_mode = N'native',
@allow_push = N'true', @allow_pull = N'true', @allow_anonymous = N'false',
@enabled_for_internet = N'false', @centralized_conflicts = N'true',
@dynamic_filters = N'false', @snapshot_in_defaultfolder = N'true',
@compress_snapshot = N'false', @ftp_port = 21, @ftp_login = N'anonymous',
@conflict_retention = 14, @keep_partition_changes = N'false',
@allow_subscription_copy = N'false', @allow_synctoalternate = N'false',
@add_to_active_directory = N'false', @max_concurrent_merge = 0,
@max_concurrent_dynamic_snapshots = 0
exec sp_addpublication_snapshot @publication = N'Sync_Log',@frequency_type
= 8, @frequency_interval = 64, @frequency_relative_interval = 0,
@frequency_recurrence_factor = 1, @frequency_subday = 1,
@frequency_subday_interval = 0, @active_start_date = 0, @active_end_date =
```

```
0, @active_start_time_of_day = 234500, @active_end_time_of_day = 0,
@snapshot_job_name = N'VM-2K-RZNW-LocExport-Sync_Log-7'
GO

exec sp_grant_publication_access @publication = N'Sync_Log', @login =
N'distributor_admin'
GO
exec sp_grant_publication_access @publication = N'Sync_Log', @login = N'sa'
GO
exec sp_grant_publication_access @publication = N'Sync_Log', @login =
N'builtin\administrators'
GO

-- Adding the merge articles
exec sp_addmergearticle @publication = N'Sync_Log', @article = N'Sync_Log',
@source_owner = N'dbo', @source_object = N'Sync_Log', @type = N'table',
@description = null, @column_tracking = N'true', @pre_creation_cmd =
N'drop', @creation_script = null, @schema_option = 0x000000000000CFF1,
@article_resolver = null, @subset_filterclause = null, @vertical_partition
= N'false', @destination_owner = N'dbo', @auto_identity_range = N'false',
@verify_resolver_signature = 0, @allow_interactive_resolver = N'false',
@fast_multicol_updateproc = N'true', @check_permissions = 0
GO
exec sp_addmergearticle @publication = N'Sync_Log', @article =
N'spMsAddSyncLogEntry', @source_owner = N'dbo', @source_object =
N'spMsAddSyncLogEntry', @type = N'proc schema only', @description = null,
@column_tracking = N'false', @pre_creation_cmd = N'drop', @creation_script
= null, @schema_option = 0x0000000000000001, @article_resolver = null,
@subset_filterclause = null, @vertical_partition = N'false',
@destination_owner = N'dbo', @auto_identity_range = N'false',
@verify_resolver_signature = 0, @destination_object =
N'spMsAddSyncLogEntry', @allow_interactive_resolver = N'false',
@fast_multicol_updateproc = N'false', @check_permissions = 0
GO
exec sp_addmergearticle @publication = N'Sync_Log', @article =
N'spMsDoReportFailedToApplyPublicationSnaphot', @source_owner = N'dbo',
@source_object = N'spMsDoReportFailedToApplyPublicationSnaphot', @type =
N'proc schema only', @description = null, @column_tracking = N'false',
@pre_creation_cmd = N'drop', @creation_script = null, @schema_option =
0x0000000000000001, @article_resolver = null, @subset_filterclause = null,
@vertical_partition = N'false', @destination_owner = N'dbo',
@auto_identity_range = N'false', @verify_resolver_signature = 0,
@destination_object = N'spMsDoReportFailedToApplyPublicationSnaphot',
@allow_interactive_resolver = N'false', @fast_multicol_updateproc =
N'false', @check_permissions = 0
GO
exec sp_addmergearticle @publication = N'Sync_Log', @article =
N'spMsAckSyncLogEntry', @source_owner = N'dbo', @source_object =
N'spMsAckSyncLogEntry', @type = N'proc schema only', @description = null,
@column_tracking = N'false', @pre_creation_cmd = N'drop', @creation_script
= null, @schema_option = 0x0000000000000001, @article_resolver = null,
@subset_filterclause = null, @vertical_partition = N'false',
@destination_owner = N'dbo', @auto_identity_range = N'false',
@verify_resolver_signature = 0, @destination_object =
N'spMsAckSyncLogEntry', @allow_interactive_resolver = N'false',
@fast_multicol_updateproc = N'false', @check_permissions = 0
GO
exec sp_addmergearticle @publication = N'Sync_Log', @article =
N'spMsDoAckAppliedPublicationSnaphot', @source_owner = N'dbo',
@source_object = N'spMsDoAckAppliedPublicationSnaphot', @type = N'proc
schema only', @description = null, @column_tracking = N'false',
@pre_creation_cmd = N'drop', @creation_script = null, @schema_option =
```

```
0x0000000000000001, @article_resolver = null, @subset_filterclause = null,
@vertical_partition = N'false', @destination_owner = N'dbo',
@auto_identity_range = N'false', @verify_resolver_signature = 0,
@destination_object = N'spMsDoAckAppliedPublicationSnaphot',
@allow_interactive_resolver = N'false', @fast_multicol_updateproc =
N'false', @check_permissions = 0
GO

-- Adding the merge subscription
exec sp_addmergesubscription @publication = N'Sync_Log', @subscriber =
N'KRONOS', @subscriber_db = N'RZNW-LocImport1', @subscription_type =
N'push', @subscriber_type = N'local', @subscription_priority = 0.000000,
@sync_type = N'automatic', @frequency_type = 4, @frequency_interval = 1,
@frequency_relative_interval = 1, @frequency_recurrence_factor = 0,
@frequency_subday = 8, @frequency_subday_interval = 1, @active_start_date =
0, @active_end_date = 0, @active_start_time_of_day = 0,
@active_end_time_of_day = 235959, @enabled_for_syncmgr = N'false',
@offloadagent = 0, @use_interactive_resolver = N'false'
GO

use [RZNW-LocExport]
GO

-- Adding the merge publication
exec sp_addmergepublication @publication = N'Notification', @description =
N'Merge publication of RZNW-Location Server''s Template Change
Notification', @retention = 14, @sync_mode = N'native', @allow_push =
N'true', @allow_pull = N'true', @allow_anonymous = N'false',
@enabled_for_internet = N'false', @centralized_conflicts = N'true',
@dynamic_filters = N'false', @snapshot_in_defaultfolder = N'true',
@compress_snapshot = N'false', @ftp_port = 21, @ftp_login = N'anonymous',
@conflict_retention = 14, @keep_partition_changes = N'false',
@allow_subscription_copy = N'false', @allow_synctoalternate = N'false',
@add_to_active_directory = N'false', @max_concurrent_merge = 0,
@max_concurrent_dynamic_snapshots = 0
exec sp_addpublication_snapshot @publication =
N'Notification',@frequency_type = 4, @frequency_interval = 1,
@frequency_relative_interval = 1, @frequency_recurrence_factor = 0,
@frequency_subday = 1, @frequency_subday_interval = 5, @active_start_date =
0, @active_end_date = 0, @active_start_time_of_day = 500,
@active_end_time_of_day = 235959, @snapshot_job_name = N'VM-2K-RZNW-
LocExport-Notification-8'
GO

exec sp_grant_publication_access @publication = N'Notification', @login =
N'distributor_admin'
GO
exec sp_grant_publication_access @publication = N'Notification', @login =
N'sa'
GO
exec sp_grant_publication_access @publication = N'Notification', @login =
N'builtin\administrators'
GO

-- Adding the merge articles
exec sp_addmergearticle @publication = N'Notification', @article =
N'LocalModifiedElements', @source_owner = N'dbo', @source_object =
N'LocalModifiedElements', @type = N'table', @description = null,
@column_tracking = N'true', @pre_creation_cmd = N'drop', @creation_script =
null, @schema_option = 0x000000000000CFF1, @article_resolver = null,
@subset_filterclause = null, @vertical_partition = N'false',
@destination_owner = N'dbo', @auto_identity_range = N'false',
```

```
@verify_resolver_signature = 0, @allow_interactive_resolver = N'false',
@fast_multicol_updateproc = N'true', @check_permissions = 0
GO

-- Adding the merge subscription
exec sp_addmergesubscription @publication = N'Notification', @subscriber =
N'KRONOS', @subscriber_db = N'RZNW-LocImport1', @subscription_type =
N'push', @subscriber_type = N'local', @subscription_priority = 0.000000,
@sync_type = N'automatic', @frequency_type = 4, @frequency_interval = 1,
@frequency_relative_interval = 1, @frequency_recurrence_factor = 0,
@frequency_subday = 8, @frequency_subday_interval = 1, @active_start_date =
0, @active_end_date = 0, @active_start_time_of_day = 0,
@active_end_time_of_day = 235959, @enabled_for_syncmgr = N'false',
@offloadagent = 0, @use_interactive_resolver = N'false'
GO
```

Anlage 4

Schnappschuss MachAdmin in die Export DB vom STAO erstellen

```
if exists (select * from dbo.sysobjects where id =
object_id('dbo.spMsMakeLocExportMachAdminSnaphot'))
  drop procedure dbo.spMsMakeLocExportMachAdminSnaphot
go

create procedure spMsMakeLocExportMachAdminSnaphot(@StartTime datetime)
as
      SET NOCOUNT ON
      declare @RC int
      declare @ErrorState int
      declare @Result integer
      declare @OwnLineIDs table([ID] uniqueidentifier)
      declare @OwnMachineIDs table([ID] uniqueidentifier)
      declare @LocationID int
      declare @TableID int

      declare @SnapshotTimestamp datetime
      declare @PublicationID char
      set @SnapshotTimestamp = GETDATE()
      set @PublicationID = 'M'

      set @Result = 0
      set @ErrorState = 0

      ------------------------------------------------------------------------
------------------------------
      -- prepare finding all data affected
      ------------------------------------------------------------------------
------------------------------

      set @LocationID = (select dbo.fnExtractOwnLocationID())
      insert into @OwnLineIDs ([ID]) select [ID] from Linie where
Standort_ID = @LocationID
      insert into @OwnMachineIDs ([ID]) select Maschine.[ID] from Maschine,
Linie where
                  (Maschine.Linie_ID = Linie.[ID]) and (Linie.Standort_ID =
@LocationID)


      BEGIN TRANSACTION
      ------------------------------------------------------------------------
------------------------------
      -- Machine Admin
      ------------------------------------------------------------------------
------------------------------

      delete from [RZNW-LocExport]..Ordner_MachAdmin
      insert into [RZNW-LocExport]..Ordner_MachAdmin
        select * from Ordner where ([LastModify] >= @StartTime)
        and (([Parent_ID] in (select Ordner_ID from TopLevelFolder where
[ID] in (4,6)))
        or ([ID] in (select Ordner_ID from TopLevelFolder where [ID] in
(4,6))) )
      if @@ERROR <> 0 begin select @ErrorState = @@ERROR goto
END_OF_PROCESSING end
```

```
      delete from [RZNW-LocExport]..Bez_MaschInfoLinie
      insert into [RZNW-LocExport]..Bez_MaschInfoLinie
        select * from Bez_MaschInfoLinie where ([LastModify] >= @StartTime)
        and ([Linie_ID] in (select * from @OwnLineIDs))
      if @@ERROR <> 0 begin select @ErrorState = @@ERROR goto
END_OF_PROCESSING end

      delete from [RZNW-LocExport]..Bez_MaschInfoMasch
      insert into [RZNW-LocExport]..Bez_MaschInfoMasch
        select * from Bez_MaschInfoMasch where ([LastModify] >= @StartTime)
        and ([Maschine_ID] in (select * from @OwnMachineIDs))
      if @@ERROR <> 0 begin select @ErrorState = @@ERROR goto
END_OF_PROCESSING end

      delete from [RZNW-LocExport]..Bez_MaschOp
      insert into [RZNW-LocExport]..Bez_MaschOp
        select * from Bez_MaschOp where ([LastModify] >= @StartTime)
        and ([Maschine_ID] in (select * from @OwnMachineIDs))
      if @@ERROR <> 0 begin select @ErrorState = @@ERROR goto
END_OF_PROCESSING end

      delete from [RZNW-LocExport]..Linie
      insert into [RZNW-LocExport]..Linie
        select * from Linie where ([LastModify] >= @StartTime)
        and ([ID] in (select * from @OwnLineIDs))
      if @@ERROR <> 0 begin select @ErrorState = @@ERROR goto
END_OF_PROCESSING end
      update [RZNW-LocExport]..Linie set [Gesperrt] = null

      delete from [RZNW-LocExport]..Maschine
      insert into [RZNW-LocExport]..Maschine
        select * from Maschine where ([LastModify] >= @StartTime)
        and ([ID] in (select * from @OwnMachineIDs))
      if @@ERROR <> 0 begin select @ErrorState = @@ERROR goto
END_OF_PROCESSING end
      update [RZNW-LocExport]..Maschine set [Gesperrt] = null

      set @TableID = (select
dbo.fnLocModEl_TableName2TableId('Alias_Namen'))
      delete from [RZNW-LocExport]..Alias_Namen_MachAdmin
      insert into [RZNW-LocExport]..Alias_Namen_MachAdmin
        select * from Alias_Namen where  ([LastModify] >= @StartTime)
        and [Alias_ID] in (select * from
dbo.fnGetLocalModifiedElementIDs(@TableID))
        and [Element_Typ] in (5, 6)
      if @@ERROR <> 0 begin select @ErrorState = @@ERROR goto
END_OF_PROCESSING end

      delete from [RZNW-LocExport]..LocalRemovedElements where Table_ID in
(15, 16, 17, 22, 23)
      insert into [RZNW-LocExport]..LocalRemovedElements
        select * from LocalRemovedElements where ([TimeStamp] >=
@StartTime)
        and (Table_ID in (15, 16, 17, 22, 23))
      if @@ERROR <> 0 begin select @ErrorState = @@ERROR goto
END_OF_PROCESSING end

--ROLLBACK
      ------------------------------------------------------------------
------------------------------
      -- set/update handshake info
```

```
      -------------------------------------------------------------------------
-----------------------------
END_OF_PROCESSING:
      if @ErrorState = 0
      begin
        COMMIT
        exec [RZNW-LocExport].dbo.spMsDoReleaseNewPublicationSnaphot
@PublicationID, @SnapshotTimestamp
      end
      else
      begin
        ROLLBACK
        set @Result = -1
      end

      return @Result

go
```

Anlage 5

Rezeptur Schnappschuss vom STAO in der Import DB am HS auf die HS DB übernehmen

```
if exists (select * from dbo.sysobjects where id =
object_id('dbo.spMsApplySnaphotTableData'))
  drop procedure dbo.spMsApplySnaphotTableData
go
create procedure dbo.spMsApplySnaphotTableData(
      @SnapshotTableName sysname = null,
      @ServerTableName sysname,
      @ServerDbName sysname,
      @IdColumnName sysname
      )
as
      declare @RC int
      declare @QueryString nvarchar(4000)
      declare @ColumnList nvarchar(4000)
      declare @ColumnUpdateList nvarchar(4000)
      declare @ServerFullTableName sysname
      declare @UnbracketedTableName sysname
      declare @ServerSysObjectsTableName sysname
      declare @LastModifyTriggerName sysname

      if @SnapshotTableName is null or @SnapshotTableName = ''
        set @SnapshotTableName = @ServerTableName

      set @ColumnList = (select
dbo.fnGetCommaSeparatedColumnList(@SnapshotTableName))
      set @ColumnUpdateList = (select
dbo.fnGetCommaSeparatedColumnUpdateList(@SnapshotTableName,
@SnapshotTableName, @ServerDbName))
      set @ServerFullTableName = @ServerDbName + '.dbo.' + @ServerTableName
      set @ServerSysObjectsTableName = @ServerDbName + '.dbo.sysobjects'

      set @UnbracketedTableName = replace(@ServerTableName, '[', '')
      set @UnbracketedTableName = replace(@UnbracketedTableName, ']', '')
      set @LastModifyTriggerName = 'tr' + @UnbracketedTableName +
'_LastModify'

      -- disable all but the LastModify triggers
      set @QueryString = 'alter table ' + @ServerFullTableName + ' disable
trigger all'
      exec @RC = sp_executesql @QueryString
      if @RC <> 0 begin return @RC end

      set @QueryString = 'if exists (select * from ' +
@ServerSysObjectsTableName +
                         ' where id = object_id(''dbo.' +
@LastModifyTriggerName + ''')) ' +
                         'alter table ' + @ServerFullTableName + ' enable
trigger ' + @LastModifyTriggerName
      exec @RC = sp_executesql @QueryString
      if @RC <> 0 begin return @RC end

      -- update existing records
      set @QueryString = 'update ' + @ServerFullTableName + ' set ' +
@ColumnUpdateList +
```

```
                        ' from ' + @SnapshotTableName + ',' +
@ServerFullTableName + ' where ' +
                        @SnapshotTableName + '.' + @IdColumnName + '=' +
@ServerFullTableName + '.' + @IdColumnName

      set @QueryString = @QueryString + ' and ' +
                        @SnapshotTableName + '.' + @IdColumnName + ' in
(select ' + @IdColumnName + ' from ' +    @ServerFullTableName + ')'
      --print convert(varchar, getdate(), 20) + ': ' + @QueryString
      exec @RC = sp_executesql @QueryString
      if @RC <> 0 begin return @RC end

      -- insert missing records
      set @QueryString = 'insert into ' + @ServerFullTableName + ' (' +
@ColumnList + ')  select ' + @ColumnList +
                        ' from ' + @SnapshotTableName + ' where ' +
@IdColumnName +
                        ' not in (select ' + @IdColumnName + ' from ' +
@ServerFullTableName + ')'
      --print convert(varchar, getdate(), 20) + ': ' + @QueryString
      exec @RC = sp_executesql @QueryString
      if @RC <> 0 begin return @RC end

      -- enable all triggers
      set @QueryString = 'alter table ' + @ServerFullTableName + ' enable
trigger all'
      exec @RC = sp_executesql @QueryString
      if @RC <> 0 begin return @RC end

      return 0
go

if exists (select * from dbo.sysobjects where id =
object_id('dbo.spMsApplySnaphotTableRemovedElements'))
  drop procedure dbo.spMsApplySnaphotTableRemovedElements
go
create procedure dbo.spMsApplySnaphotTableRemovedElements(
      @ServerTableName sysname,
      @ServerDbName sysname,
      @IdColumnName sysname,
      @AdditionalElementsToBeRemovedFilter varchar(1000) = ''
      )
as
      declare @RC int
      declare @QueryString nvarchar(1000)
      declare @ServerFullTableName sysname
      declare @TableID int
      declare @SqlParameters nvarchar(1000)

      set @ServerFullTableName = @ServerDbName + '.dbo.' + @ServerTableName

      -- delete removed records
      set @QueryString = N'set @TableID = ' + @ServerDbName +
'.dbo.fnLocModEl_TableName2TableId(''' + @ServerTableName + ''')'
      set @SqlParameters = N'@TableID int OUTPUT'
      execute @RC = sp_executesql @QueryString, @SqlParameters, @TableID
OUTPUT
      if @RC <> 0 begin return @RC end
      set @QueryString = 'delete from ' + @ServerFullTableName + ' where '
+
```

```
                        @IdColumnName + ' in (select Element_ID from
LocalRemovedElements where Table_ID = ' +       convert(varchar, @TableID) +
')'
      if not @AdditionalElementsToBeRemovedFilter is null and
@AdditionalElementsToBeRemovedFilter <> ''
        set @QueryString = @QueryString + ' or ' +
@AdditionalElementsToBeRemovedFilter

      --print convert(varchar, getdate(), 20) + ': ' + @QueryString
      exec @RC = sp_executesql @QueryString
      if @RC <> 0 begin return @RC end

      return 0
go


if exists (select * from dbo.sysobjects where id =
object_id('dbo.spMsApplyLocExportRecipeSnaphot'))
  drop procedure dbo.spMsApplyLocExportRecipeSnaphot
go

create procedure spMsApplyLocExportRecipeSnaphot(
      @ServerDbName sysname
      )
as
      SET NOCOUNT ON
      declare @RC int
      declare @Result integer
      declare @SyncLogEntryID uniqueidentifier
      declare @PublicationID char
      declare @SyncTimeStampInfo datetime
      declare @PubTimeStampInfo datetime
      declare @OldestTimeStamp datetime
      declare @ErrorState int
      declare @RecipeDependenciesToBeRemovedFilter varchar(1000)

      set @Result = 0
      set @ErrorState = 0

      if @ServerDbName is null or @ServerDbName = ''
        return -2
      set @PublicationID = 'R'

      set @SyncLogEntryID = (select top 1 [ID] from Sync_Log where [Event]
= 1 and [Ack]=0  and [Publication]=@PublicationID order by TimeStamp_UTC
desc)
      set @SyncTimeStampInfo = (select [SyncTimeStampInfo] from Sync_Log
where [ID] = @SyncLogEntryID)
      set @PubTimeStampInfo = (select [PubTimeStamp] from SnapshotInfo
where Publication = @PublicationID)
      if @PubTimeStampInfo is null or @SyncTimeStampInfo is null or
@SyncTimeStampInfo <> @PubTimeStampInfo
        return -100

      set @RecipeDependenciesToBeRemovedFilter = '(Rezeptur_ID in (select
Element_ID from LocalRemovedElements where Table_ID = 25))'

      -- insert/update
      BEGIN TRANSACTION
      exec @RC = spMsApplySnaphotTableData
@SnapshotTableName='[Ordner_Rcp]', @ServerTableName='[Ordner]',
@ServerDbName=@ServerDbName, @IdColumnName='[ID]'
```

```
      if @RC <> 0 begin select @ErrorState = @RC goto END_OF_PROCESSING end
      exec @RC = spMsApplySnaphotTableData @ServerTableName='[Rezeptur]',
@ServerDbName=@ServerDbName, @IdColumnName='[ID]'
      if @RC <> 0 begin select @ErrorState = @RC goto END_OF_PROCESSING end

      exec @RC = spMsApplySnaphotTableData
@SnapshotTableName='[Bez_MerkmalOrdner_Rcp]',
@ServerTableName='[Bez_MerkmalOrdner]', @ServerDbName=@ServerDbName,
@IdColumnName='[ID]'
      if @RC <> 0 begin select @ErrorState = @RC goto END_OF_PROCESSING end
      exec @RC = spMsApplySnaphotTableData
@ServerTableName='[Bez_RezMerkmal]', @ServerDbName=@ServerDbName,
@IdColumnName='[ID]'
      if @RC <> 0 begin select @ErrorState = @RC goto END_OF_PROCESSING end
      exec @RC = spMsApplySnaphotTableData
@ServerTableName='[Bez_RezOpMasch]', @ServerDbName=@ServerDbName,
@IdColumnName='[ID]'
      if @RC <> 0 begin select @ErrorState = @RC goto END_OF_PROCESSING end
      exec @RC = spMsApplySnaphotTableData @ServerTableName='[Bez_RezPrm]',
@ServerDbName=@ServerDbName, @IdColumnName='[ID]'
      if @RC <> 0 begin select @ErrorState = @RC goto END_OF_PROCESSING end
      exec @RC = spMsApplySnaphotTableData
@ServerTableName='[Bez_RezZsMasch]', @ServerDbName=@ServerDbName,
@IdColumnName='[ID]'
      if @RC <> 0 begin select @ErrorState = @RC goto END_OF_PROCESSING end
      exec @RC = spMsApplySnaphotTableData
@ServerTableName='[Parameter_Blob]', @ServerDbName=@ServerDbName,
@IdColumnName='[ID]'
      if @RC <> 0 begin select @ErrorState = @RC goto END_OF_PROCESSING end
      -- do not apply maschine prm from stao on hs
      --exec @RC = spMsApplySnaphotTableData
@ServerTableName='[Bez_MaschPrmPrm]', @ServerDbName=@ServerDbName,
@IdColumnName='[ID]'
      --if @RC <> 0 begin select @ErrorState = @RC goto END_OF_PROCESSING
end

      -- remove in reverse order
      --exec @RC = spMsApplySnaphotTableRemovedElements
@ServerTableName='[Bez_MaschPrmPrm]', @ServerDbName=@ServerDbName,
@IdColumnName='[ID]'
      --if @RC <> 0 begin select @ErrorState = @RC goto END_OF_PROCESSING
end
      --exec @RC = spMsApplySnaphotTableRemovedElements
@ServerTableName='[Parameter_Blob]', @ServerDbName=@ServerDbName,
@IdColumnName='[ID]'
      if @RC <> 0 begin select @ErrorState = @RC goto END_OF_PROCESSING end
      exec @RC = spMsApplySnaphotTableRemovedElements
@ServerTableName='[Bez_RezZsMasch]', @ServerDbName=@ServerDbName,
@IdColumnName='[ID]',

      @AdditionalElementsToBeRemovedFilter=@RecipeDependenciesToBeRemovedFi
lter
      if @RC <> 0 begin select @ErrorState = @RC goto END_OF_PROCESSING end
      exec @RC = spMsApplySnaphotTableRemovedElements
@ServerTableName='[Bez_RezPrm]', @ServerDbName=@ServerDbName,
@IdColumnName='[ID]',

      @AdditionalElementsToBeRemovedFilter=@RecipeDependenciesToBeRemovedFi
lter
      if @RC <> 0 begin select @ErrorState = @RC goto END_OF_PROCESSING end
```

```
      exec @RC = spMsApplySnaphotTableRemovedElements
@ServerTableName='[Bez_RezOpMasch]', @ServerDbName=@ServerDbName,
@IdColumnName='[ID]',

      @AdditionalElementsToBeRemovedFilter=@RecipeDependenciesToBeRemovedFi
lter
      if @RC <> 0 begin select @ErrorState = @RC goto END_OF_PROCESSING end
      exec @RC = spMsApplySnaphotTableRemovedElements
@ServerTableName='[Bez_RezMerkmal]', @ServerDbName=@ServerDbName,
@IdColumnName='[ID]',

      @AdditionalElementsToBeRemovedFilter=@RecipeDependenciesToBeRemovedFi
lter
      if @RC <> 0 begin select @ErrorState = @RC goto END_OF_PROCESSING end
      exec @RC = spMsApplySnaphotTableRemovedElements
@ServerTableName='[Bez_MerkmalOrdner]', @ServerDbName=@ServerDbName,
@IdColumnName='[ID]'
      if @RC <> 0 begin select @ErrorState = @RC goto END_OF_PROCESSING end

      exec @RC = spMsApplySnaphotTableRemovedElements
@ServerTableName='[Rezeptur]', @ServerDbName=@ServerDbName,
@IdColumnName='[ID]'
      if @RC <> 0 begin select @ErrorState = @RC goto END_OF_PROCESSING end
      exec @RC = spMsApplySnaphotTableRemovedElements
@ServerTableName='[Ordner]', @ServerDbName=@ServerDbName,
@IdColumnName='[ID]'
      if @RC <> 0 begin select @ErrorState = @RC goto END_OF_PROCESSING end

--ROLLBACK TRANSACTION
      ------------------------------------------------------------------------
------------------------------
      -- set/update handshake info
      ------------------------------------------------------------------------
------------------------------
END_OF_PROCESSING:
      if @ErrorState = 0
      begin
        COMMIT
        exec dbo.spMsDoAckAppliedPublicationSnaphot
@SynLogSnaphotEntryID=@SyncLogEntryID,
                  @PublicationID=@PublicationID,
@SnapshotTimestamp=@SyncTimeStampInfo
      end
      else
      begin
        ROLLBACK
        exec dbo.spMsDoReportFailedToApplyPublicationSnaphot
@SynLogSnaphotEntryID=@SyncLogEntryID,
                  @PublicationID=@PublicationID,
@SnapshotTimestamp=@SyncTimeStampInfo
        set @Result = -1
      end

      return @Result
go

if exists (select * from dbo.sysobjects where id =
object_id('dbo.spMsApplyLocExportMachAdminSnaphot'))
  drop procedure dbo.spMsApplyLocExportMachAdminSnaphot
go

create procedure spMsApplyLocExportMachAdminSnaphot(
```

```
	@ServerDbName sysname
	)
as
	SET NOCOUNT ON
	declare @RC int
	declare @Result integer
	declare @SyncLogEntryID uniqueidentifier
	declare @PublicationID char
	declare @SyncTimeStampInfo datetime
	declare @PubTimeStampInfo datetime
	declare @OldestTimeStamp datetime
	declare @ErrorState int
	declare @MachineDependenciesToBeRemovedFilter varchar(1000)
	declare @LineDependenciesToBeRemovedFilter varchar(1000)

	set @Result = 0
	set @ErrorState = 0

	if @ServerDbName is null or @ServerDbName = ''
	  return -2
	set @PublicationID = 'M'

	set @SyncLogEntryID = (select top 1 [ID] from Sync_Log where [Event]
= 1 and [Ack]=0  and [Publication]=@PublicationID order by TimeStamp_UTC
desc)
	set @SyncTimeStampInfo = (select [SyncTimeStampInfo] from Sync_Log
where [ID] = @SyncLogEntryID)
	set @PubTimeStampInfo = (select [PubTimeStamp] from SnapshotInfo
where Publication = @PublicationID)
	if @PubTimeStampInfo is null or @SyncTimeStampInfo is null or
@SyncTimeStampInfo <> @PubTimeStampInfo
	  return -100

	set @MachineDependenciesToBeRemovedFilter = '(Maschine_ID in (select
Element_ID from LocalRemovedElements where Table_ID = 23))'
	set @LineDependenciesToBeRemovedFilter = '(Linie_ID in (select
Element_ID from LocalRemovedElements where Table_ID = 22))'

	-- insert/update
	BEGIN TRANSACTION
	exec @RC = spMsApplySnaphotTableData
@SnapshotTableName='[Ordner_MachAdmin]', @ServerTableName='[Ordner]',
@ServerDbName=@ServerDbName, @IdColumnName='[ID]'
	if @RC <> 0 begin select @ErrorState = @RC goto END_OF_PROCESSING end
	exec @RC = spMsApplySnaphotTableData @ServerTableName='[Linie]',
@ServerDbName=@ServerDbName, @IdColumnName='[ID]'
	if @RC <> 0 begin select @ErrorState = @RC goto END_OF_PROCESSING end
	exec @RC = spMsApplySnaphotTableData @ServerTableName='[Maschine]',
@ServerDbName=@ServerDbName, @IdColumnName='[ID]'
	if @RC <> 0 begin select @ErrorState = @RC goto END_OF_PROCESSING end
	exec @RC = spMsApplySnaphotTableData
@ServerTableName='[Bez_MaschInfoLinie]', @ServerDbName=@ServerDbName,
@IdColumnName='[ID]'
	if @RC <> 0 begin select @ErrorState = @RC goto END_OF_PROCESSING end
	exec @RC = spMsApplySnaphotTableData
@ServerTableName='[Bez_MaschInfoMasch]', @ServerDbName=@ServerDbName,
@IdColumnName='[ID]'
	if @RC <> 0 begin select @ErrorState = @RC goto END_OF_PROCESSING end
	exec @RC = spMsApplySnaphotTableData
@ServerTableName='[Bez_MaschOp]', @ServerDbName=@ServerDbName,
@IdColumnName='[ID]'
	if @RC <> 0 begin select @ErrorState = @RC goto END_OF_PROCESSING end
```

```
      exec @RC = spMsApplySnaphotTableData
@SnapshotTableName='[Alias_Namen_MachAdmin]',
@ServerTableName='[Alias_Namen]', @ServerDbName=@ServerDbName,
@IdColumnName='[Alias_ID]'
      if @RC <> 0 begin select @ErrorState = @RC goto END_OF_PROCESSING end

      -- remove in reverse order
      exec @RC = spMsApplySnaphotTableRemovedElements
@ServerTableName='[Bez_MaschOp]', @ServerDbName=@ServerDbName,
@IdColumnName='[ID]',

      @AdditionalElementsToBeRemovedFilter=@MachineDependenciesToBeRemovedF
ilter
      if @RC <> 0 begin select @ErrorState = @RC goto END_OF_PROCESSING end
      exec @RC = spMsApplySnaphotTableRemovedElements
@ServerTableName='[Bez_MaschInfoMasch]', @ServerDbName=@ServerDbName,
@IdColumnName='[ID]',

      @AdditionalElementsToBeRemovedFilter=@MachineDependenciesToBeRemovedF
ilter
      if @RC <> 0 begin select @ErrorState = @RC goto END_OF_PROCESSING end
      exec @RC = spMsApplySnaphotTableRemovedElements
@ServerTableName='[Bez_MaschInfoLinie]', @ServerDbName=@ServerDbName,
@IdColumnName='[ID]',

      @AdditionalElementsToBeRemovedFilter=@LineDependenciesToBeRemovedFilt
er
      if @RC <> 0 begin select @ErrorState = @RC goto END_OF_PROCESSING end
      exec @RC = spMsApplySnaphotTableRemovedElements
@ServerTableName='[Maschine]', @ServerDbName=@ServerDbName,
@IdColumnName='[ID]'
      if @RC <> 0 begin select @ErrorState = @RC goto END_OF_PROCESSING end
      exec @RC = spMsApplySnaphotTableRemovedElements
@ServerTableName='[Linie]', @ServerDbName=@ServerDbName,
@IdColumnName='[ID]'
      if @RC <> 0 begin select @ErrorState = @RC goto END_OF_PROCESSING end
      --exec @RC = spMsApplySnaphotTableRemovedElements
@ServerTableName='[Ordner]', @ServerDbName=@ServerDbName,
@IdColumnName='[ID]'
      --if @RC <> 0 begin select @ErrorState = @RC goto END_OF_PROCESSING
end

--ROLLBACK TRANSACTION
      ------------------------------------------------------------------------
------------------------------
      -- set/update handshake info
      ------------------------------------------------------------------------
------------------------------
END_OF_PROCESSING:
      if @ErrorState = 0
      begin
        COMMIT
        exec dbo.spMsDoAckAppliedPublicationSnaphot
@SynLogSnaphotEntryID=@SyncLogEntryID,
                  @PublicationID=@PublicationID,
@SnapshotTimestamp=@SyncTimeStampInfo
      end
      else
      begin
        ROLLBACK
        exec dbo.spMsDoReportFailedToApplyPublicationSnaphot
@SynLogSnaphotEntryID=@SyncLogEntryID,
```

```
                  @PublicationID=@PublicationID,
@SnapshotTimestamp=@SyncTimeStampInfo
        set @Result = -1
      end

      return @Result
go
```

Anlage 6

Bsp: Trigger auf der Tabelle Merkmal um die Änderungen in Logtabellen zu protokollieren.

```
if exists (select * from dbo.sysobjects where id =
object_id('dbo.trMerkmal_Modification'))
drop trigger dbo.trMerkmal_Modification
go
create trigger dbo.trMerkmal_Modification
on Merkmal
for update, insert, delete as
  if @@ROWCOUNT=0 return
  if not exists(select * from deleted) or
      not (update(Timestamp_UTC) and not update(ID)) and not
update(LastModify)
        and not ([dbo].fnDbStateGetTemplateLoggingDisabled() <> 0)
  begin
    declare @ModifiedTableID int
    declare @TableName nvarchar (50)
    ---------------------------------------------
    set @ModifiedTableID = 2
    set @TableName = 'Merkmal'
    ---------------------------------------------

    declare @Action char
    declare @RowIDs table([ID] uniqueidentifier)
    declare @User nvarchar (50)
    declare @Removed bit

    set @User = dbo.fnGetTemplateLockingUser()
    if @User is null or @User = ''
      set @User = dbo.fnGetMachineParameterLockingUser()
    if exists(select * from inserted)
    begin
      insert into @RowIDs ([ID]) select [ID] from inserted
      set @Removed = 0
      if exists(select * from deleted)
        set @Action = 'U'     -- update
      else
        set @Action = 'I'     -- insert
    end
    else
    begin
      insert into @RowIDs ([ID]) select [ID] from deleted
      set @Action = 'D'       -- delete
      set @Removed = 1
    end

    if @Action = 'U'
    begin
      select * into #tmp_inserted from inserted
      select * into #tmp_deleted from deleted
      declare     @TablesAreEqual bit
      exec dbo.spCompareTableContentsForEquality
      @TableA = '#tmp_inserted',
      @TableB = '#tmp_deleted',
      @ColumnListSrcTable = @TableName,
      @TablesAreEqual = @TablesAreEqual OUTPUT
      if @TablesAreEqual <> 0
```

```
        return
    end

    update [dbo].[LocalModifiedElements] set [Modified_UTC] = GETUTCDATE(),
[ModifyingUser] = @User, [Removed] = @Removed where
                    ([Table_ID] = @ModifiedTableID) and ([Element_ID] in
(select [ID] from @RowIDs))

    insert into [dbo].[LocalModifiedElements] ([Table_ID], [Element_ID],
[Modified_UTC], [ModifyingUser], [Snapshot_UTC], [Removed])
                    select @ModifiedTableID, [ID], GETUTCDATE(), @User, null,
@Removed from @RowIDs where
                    ([ID] not in (select [Element_ID] from
[dbo].[LocalModifiedElements] where [Table_ID] = @ModifiedTableID))

    if (@Action <> 'U') and not
([dbo].fnDbStateGetTemplateChangeLogDisabled() <> 0)
      insert into [dbo].[AutoLog_Templates] ([TimeStamp], [TableName],
[Element_ID], [Action], [UserName], [Origin_Loc_ID])
                    select GETDATE(), @TableName, [ID], @Action, @User, null
from @RowIDs
  end
go
```

Literaturverzeichnis

/TKT-Presta Info/ ThyssenKrupp Presta Camshafts, WWW: http://de.wikipedia.org/w/index.php?oldid=76651616 verfügbar am 12.05.2011

[Skulschus 07] Skulschus, M. (2007). SQL und relationale Datenbanken, Verlag Comelio Medien, Essen, ISBN 3-939701-11-4

[Schicker 96] Schicker, E. (1996), Datenbanken und SQL Eine praxisorientierte Einführung, Verlag Teubner, Stuttgart 1996, ISBN 3-519-02991-X

[Klein 05] Kleinschmidt, P., Rank C. (2005), Relationale Datenbanksysteme – Eine praktische Einführung, Verlag Springer, 2. Aufl. Berlin, ISBN 3-540-42413-X

[Cordts 02] Cordts, S. (2002), Datenbankkonzepte In der Praxis nach dem Standard SQL-99, Verlag Addison Wesley, München, ISBN 3-8273-1938-2

[Saake 03] Saake, G., Schmitt, I., Türker, C. (2003), Objektdatenbanken - Konzepte, Sprachen, Architekturen, Verlag International Thomson Publishing, Magdeburg, ISBN 3-8266-0258-7 PDF Version

[Wieken 09] Wieken, J. (2009), SQL – Einstieg für Anspruchsvolle, Verlag Addison Wesley, München, ISBN 978-3-8273-2485-6

[Codd 70] Codd, E.F. (1970), www: http://www.seas.upenn.edu/~zives/03f/cis550/codd.pdf Verfügbar 12.05.2011

[Matt 09] Matteschеck, S. (2009), SQL Server 2008 – Das Programmierhandbuch, Verlag Galileo Press, Bonn, 3. aktualisierte und erweiterte Auflage, ISBN 978-3-8362-1395-0

[Hor 00] Hornung, H., (2000), Replikation in verteilten Datenbanken, Fachbereich Rechts- und Wirtschaftswissenschaften, Technische Universität Darmstadt, hornung@rbg.informatik.tu-darmstadt.de

[Bern 87] Bernstein, P. A., Hadzilacos, V., Goodman, N.: Concurrency Control and Recovery in Database Systems, Verlag Addison Wesley, New York 1987

[Gray 96] Gray, J., Helland, P., O'Neil, P. ,(1996), 'The Dangers of Replication and a

Solution“, in: Proc. of the SIGMOD Conf, p.173-182.

[Mutsch 04] Mutschler, B., Specht, G., (2004), Mobile Datenbanksysteme - Architektur, Implementierung, Konzepte, Verlag Springer, Berlin, ISBN 978-3-540-20886-0

[Munz 04] Munz, F., (2004), Einsatz von Replikation, Technische Universität Kaiserslautern SS 2004, f_munz@informatik.uni-kl.de, 16. Juli 2004

[Tiem 09] Tiemeyer, E. (2009), Handbuch IT-Management: Konzepte, Methoden, Lösungen und Arbeitshilfen für die Praxis, Verlag Hanser, 3. Aufl., ISBN 978-3446418424

[Beut 96] Beuter, T., Dadam, P. (1996), Prinzipien der Replikationskontrolle in verteilten Datenbanksystemen, Informatik Forsch. Entw. 11: p 203–212, Abteilung Datenbanken und Informationssysteme, Fakultötat fü Informatik, Universität Ulm, WWW: http://www.informatik.uni-ulm.de/abt/dbis

/MSRep 11/ MSDN SQL Srv 2008 (2011), Das Replikationsveröffentlichungsmodell, WWW: http://msdn.microsoft.com/de-de/library/ms152567%28v=SQL.105%29.aspx,verfügbar 22.05.2011

/Repl 11/ MSDN SQL Srv 2008 (2011), Replikationsübersicht, WWW: http://msdn.microsoft.com/de-de/library/bb500346.aspx, verfügbar 22.05.2011

/ReTy 11/ MSDN SQL Srv 2008 (2011), Replikationstypen, WWW: http://msdn.microsoft.com/de-de/library/ms152531.aspx, verfügbar 22.05.2011

/ReTa 11/ MSDN SQL Srv 2008 (2011), Funktionsweise der Transaktionsreplikation, WWW: http://msdn.microsoft.com/de-de/library/ms151706.aspx, verfügbar 22.05.2011

/ReMr 11/ MSDN SQL Srv 2008 (2011), Funktionsweise der Mergereplikation, WWW: http://msdn.microsoft.com/de-de/library/ms151329.aspx, verfügbar 22.05.2011

/ReSr 11/ MSDN SQL Srv 2008 (2011), Funktionsweise der Snapshotreplikation, WWW: http://msdn.microsoft.com/de-de/library/ms151734.aspx, verfügbar 22.05.2011

[CeSm 05] J Celko, J., (2005), Sql For Smarties - Advanced Sql Programming, Verlag

Morgan Kaufmann, San Francisco, 3rd Edt, ISBN 0-12-369379-9

[Hend 01] Henderson, K., (2001), Guru's Guide to SQL Server Stored Procedures, XML, and HTML, Verlag Addison Wesley, ISBN 0-201-70046-8

[CePu 07] J Celko, J., (2007), Sql Puzzles And Answers, Verlag Morgan Kaufmann, San Francisco, 2nd Edt, ISBN 0-12-373596-3

[CeTree 04] J Celko, J., (2004), Trees And Hierarchies In Sql For Smarties, Verlag Morgan Kaufmann, San Francisco, 2nd Edt, ISBN 1-55860-920-2

/Codd 85/ Codd, E. (1985), Die 12 goldenen Regeln für relationale Datenbanken, WWW: http://www.sk-consulting.de/www/html-data/infodb2/DB12Regeln.htm ,verfügbar 26.05.2011

Erklärung

Hiermit erkläre ich, dass ich die vorliegende Arbeit selbständig und nur unter Verwendung der angegebenen Literatur und Hilfsmittel angefertigt habe.

Stellen, die aus Quellen entnommen wurden, sind als solche kenntlich gemacht.

Diese Arbeit wurde in gleicher oder ähnlicher Form noch keiner anderen Prüfungsbehörde vorgelegt.

Lustenau, den 26. Mai 2011