



JOHANN WOLFGANG GOETHE – UNIVERSITÄT  
FRANKFURT AM MAIN

---

Fachbereich Informatik und Biologie  
Institut für Informatik  
Datenbanken und Informationssysteme

**Performance Optimierung der Nutzung  
von RDF-S3 Daten am Beispiel  
von IBMs DB2 v8.1**

Diplomarbeit  
von  
**Jashar Rexhepi**

Erstprüfer: Prof. Dott.-Ing. Roberto V. Zicari

Betreuer: Dipl.-Math. Karsten Tolle

14. März 2005



Hiermit erkläre ich, dass ich die vorliegende Diplomarbeit selbständig verfasst und keine anderen als die angegebenen Quellen, Hilfsmittel und Geräte benutzt habe.

Offenbach am Main, 14. März 2005

Jashar Rexhepi



# Inhaltsverzeichnis

<b>1</b>	<b>Allgemeines</b>	<b>9</b>
1.1	Motivation . . . . .	9
1.2	Zielsetzung . . . . .	10
1.3	Gliederung . . . . .	10
1.4	Danksagungen . . . . .	11
<b>2</b>	<b>Grundlagen</b>	<b>13</b>
2.1	Einführung in die Optimierung . . . . .	13
2.1.1	Warum optimieren? . . . . .	13
2.2	RDF-Source related Storage System . . . . .	15
2.3	Das Datenbanksystem . . . . .	18
2.3.1	Architektur und Prozesse . . . . .	18
2.3.2	Übersicht über Speicherobjekte . . . . .	21
2.3.3	Tabellen und Indizes . . . . .	25
2.3.4	Speicherverwaltung . . . . .	28
<b>3</b>	<b>Optimierung durch Vergleichstests</b>	<b>31</b>
3.1	Vergleichstestmethoden . . . . .	31
3.2	Ausführung von Vergleichstests – Vorgehensweise . . . . .	36
3.3	Die Testumgebung . . . . .	37
<b>4</b>	<b>Optimierung der Konfigurationsparameter</b>	<b>39</b>
4.1	Die Testdaten und Methoden . . . . .	39
4.1.1	Testmethoden . . . . .	40
4.2	RDF-S3 Entwicklung . . . . .	43
4.3	SMS vs. DMS . . . . .	45
4.4	Die Tabellenbereiche . . . . .	47
4.4.1	Die Konfiguration der Tabellenbereiche . . . . .	48
4.5	Datenverteilung auf mehrere Tabellenbereiche . . . . .	49
4.6	Der Pufferpool . . . . .	51
4.6.1	Direkte Veränderungen am Pufferpool . . . . .	53
4.6.2	Die Veränderung an den abhängigen Parametern . . . . .	56
4.7	Das Logging . . . . .	56

4.8	Die Parallelisierung . . . . .	58
4.9	Konfiguration anderer Parameter . . . . .	59
4.10	Die optimierte Datenbank . . . . .	60
4.11	Schema-Daten . . . . .	63
4.12	Stream Based Parsing . . . . .	64
<b>5</b>	<b>Zusammenfassung und Ausblick</b>	<b>67</b>
5.1	Zusammenfassung . . . . .	67
5.2	Empfehlung . . . . .	68
5.3	Ausblick . . . . .	69
<b>A</b>	<b>DB2 Konfigurationsparameter</b>	<b>71</b>
<b>B</b>	<b>Inhaltsbeschreibung der CD</b>	<b>73</b>

# Abbildungsverzeichnis

2.1	Screenshot von RDF-S3. . . . .	16
2.2	Die Arbeitsweise von RDF-S3. . . . .	17
2.3	Architektur- und Prozessübersicht [DB2-02] . . . . .	19
2.4	Tabellenbereiche und Tabellen in einer Datenbank. . . . .	22
2.5	Übersicht über Tabellenbereiche. . . . .	23
2.6	SMS- und DMS-Tabellenbereiche. . . . .	24
2.7	Logische Tabellen-, Datensatz und Indexstruktur für Standardtabellen. . . . .	26
2.8	Logische Tabellen-, Datensatz- und Indexstruktur für MDC-Tabellen. . . . .	27
2.9	Arten und Verwendung des Speichers durch den Datenbankmanager. . . . .	28
3.1	Screenshot von Visual Explain. . . . .	33
3.2	Die Ausgabe von RDF-S3 (Screenshot). . . . .	35
4.1	Vergleich der beiden Testmethoden. . . . .	41
4.2	Test mit RDF-Dateien kleiner Größe und SMS als Tabellenbereich. . . . .	42
4.3	Auswirkungen einer Hardwareaufrüstung (Festplatte) auf die Laufzeiten. . . . .	43
4.4	Vergleich zwischen verschiedenen Versionen von RDF-S3. . . . .	44
4.5	Die Anpassung des Wertes der <i>Cache-Resources</i> Variable. . . . .	45
4.6	Der Vergleich von SMS mit DMS – kleine Dateien. . . . .	46
4.7	Der Vergleich von SMS mit DMS – mittelgroße Dateien. . . . .	47
4.8	Der Vergleich von SMS mit DMS – große Dateien. . . . .	47
4.9	Konfigurationsparameter <i>prefetchsize</i> und <i>extentsize</i> . . . . .	49
4.10	Trennung der Tabellen <i>PROPERTIES</i> und <i>RESOURCES</i> . . . . .	51
4.11	Screenshot einer Momentaufnahme während eines Tests mit mittelgroßen Dateien. Aus den Angaben ergibt sich ein Wert von Hit Ratio = 72 %. . . . .	53
4.12	Datenbankkonfiguration mit unterschiedlicher Pufferpoolanzahl und -Größe. – kleine Dateien. . . . .	54
4.13	Datenbankkonfiguration mit unterschiedlicher Pufferpoolanzahl und -Größe. – mittelgroße Dateien. . . . .	54
4.14	Datenbankkonfiguration mit unterschiedlicher Pufferpoolanzahl und -Größe. – große Dateien. . . . .	55
4.15	Optimierung der Protokolldateianzahl und -Größe. . . . .	58
4.16	Auswirkungen eines erhöhten Wertes für <i>logbufsiz</i> -Parameter. . . . .	59

4.17 Standardkonfiguration vs. optimierte Konfiguration – kleine Dateien. . . . .	62
4.18 Standardkonfiguration vs. optimierte Konfiguration – mittelgroße Dateien. . . . .	62
4.19 Standardkonfiguration vs. optimierte Konfiguration – große Dateien. . . . .	63
4.20 Schema-Daten – Standardkonfiguration vs. optimierte Konfiguration. . . . .	64
4.21 Standardkonfiguration vs. optimierte Konfiguration (kleine Dateien) – <i>Stream Based Parsing</i> -Methode. . . . .	65
4.22 Standardkonfiguration vs. optimierte Konfiguration (mittelgroße Dateien) – <i>Stream Based Parsing</i> -Methode. . . . .	66
4.23 Standardkonfiguration vs. optimierte Konfiguration (große Dateien) – <i>Stream Based Parsing</i> -Methode. . . . .	66

# Kapitel 1

## Allgemeines

### 1.1 Motivation

Das *World Wide Web* (WWW) umfasst eine gewaltige Menge unterschiedlicher Informationen, die von vielen Organisationen, Gemeinschaften und Einzelpersonen, aus welchen Gründen auch immer, erzeugt wurden. Die Benutzer des Webs können auf verschiedene Weise und vor allem sehr leicht auf diese Informationen zugreifen, wie z.B. durch die Angabe der Adresse (URL), Suchanfragen usw. Als Folge dieses ungebremsen Wachstums des Webs, existieren mittlerweile in vielen Anwendungen riesige Mengen an Metadaten (Daten über Daten) und es gestaltet sich immer schwieriger Informationen zu suchen, zu organisieren und zu pflegen. Das Filtern der gewünschten von ungewünschten (irrelevanten) Informationen benötigt manchmal sehr viel Geduld und Geschick.

Zur Lösung dieser Probleme entstand die Idee des *semantischen Webs*<sup>1</sup> (Semantic Web) als eine Erweiterung des gegenwärtigen WWW, in dem Daten eine wohldefinierte, maschinenverarbeitbare Bedeutung (Semantik) gegeben wird. Das semantische Web verfolgt die Idee, Daten im Web derart zu beschreiben, dass sie sowohl von automatisierten Werkzeugen als auch von Menschen geteilt und verarbeitet werden können<sup>2</sup>. Diese Weiterentwicklung des WWW wird u.a. von der Initiative *Semantic Web* des W3C mithilfe des *Resource Description Frameworks* (RDF)<sup>3</sup> verfolgt. RDF hat zum Ziel, die Erzeugung und den Austausch von Metadaten so einfach wie möglich zu machen. RDF-Daten können als eine Menge von einfachen Aussagen betrachtet werden, wobei jede Aussage aus *Subjekt*, *Prädikat* und *Objekt* besteht, die als *Tripel* bezeichnet werden.

In der heutigen Zeit ist das semantische Web noch nicht etabliert. Um die Verbreitung des semantischen Webs voranzutreiben, werden RDF-Datenverwaltungssysteme, basierend

---

<sup>1</sup><http://www.w3.org/2001/sw/>

<sup>2</sup>“The Semantic Web is an extension of the current web in which information is given well-defined meaning, better enabling computers and people to work in cooperation.“ – Tim Berners-Lee, James Hendler, Ora Lassila, The Semantic Web, Scientific American, May 2001.

<sup>3</sup><http://www.w3.org/RDF/>

auf existierende bewährte Systeme wie (O)RDBMS und SQL, benötigt. *RDF Source related Storage System* (RDF-S3) (Abbildung 2.1), Gegenstand dieser Arbeit, ist eine solche Anwendung und kann online unter:

<http://www.dbis.cs.uni-frankfurt.de/~tolle/RDF/RDFS3/>

bezogen werden.

## 1.2 Zielsetzung

Das Ziel dieser Diplomarbeit ist die Performance-Optimierung der Anwendung RDF-S3. Die Aufgabe der Anwendung RDF-S3 ist das Speichern von RDF-Daten. Dies wird am Beispiel von IBM's DB2 v8.1 als Datenbankverwaltungssystem durchgeführt. Dabei soll die Optimierung (Verbesserung) der Datenbankkonfigurationsparameter eine zentrale Rolle spielen.

Die Erreichung eines Ziels bei der Optimierung sollte auch durch Messungen beweisbar und nachvollziehbar sein.

Als Hauptkriterium der Optimierung dient die Antwortzeit d.h. wie lange dauert es, bis eine Operation oder eine Gruppe von Operationen im Durchschnitt ausgeführt werden. Bei dieser Arbeit sind das die INSERT-Operationen.

Durch die Optimierung der Konfigurationsparameter soll sich am Ende dieser Diplomarbeit das Laufzeitverhalten von RDF-S3 deutlich verbessern. Das Ziel ist nicht durch eine quantitative Festlegung definierbar. Demnach, es ist nicht festgelegt, dass die Performance von RDF-S3 z.B. um 10 % oder 20 % steigen soll.

## 1.3 Gliederung

Im ersten Teil dieser Arbeit wird dem Leser zunächst die Software vorgestellt. Danach folgt ein großer Abschnitt, in dem der eigentliche Optimierungsvorgang dargestellt wird. Die Ergebnisse werden jeweils in kurzen Abschnitten besprochen und präsentiert.

**Kapitel 2** präsentiert zuerst einen kurzen Überblick über die grundlegenden Kriterien des Optimierungsprozesses. Dann wird die Anwendung RDF-S3 kurz vorgestellt. Im Anschluss wird das verwendete Datenbankmanagementsystem mit seinen wichtigsten Komponenten vorgestellt.

**Kapitel 3** beschäftigt sich näher mit den Vergleichstestmethoden bei der Optimierung. Zudem werden viele Werkzeuge, die bei der Optimierung und Analyse hilfreich sein können, vorgestellt. Danach wird die Standarddatenbank, die als Grundlage für die Tests dient, präsentiert, um anschließend die Testumgebung vorzustellen.

**Kapitel 4** beschäftigt sich mit der Konfiguration der Datenbank. Dabei werden die wichtigsten Optimierungsparameter näher betrachtet. Die gewonnenen Ergebnisse aus den

Tests werden in kleinen Abschnitten präsentiert. Der zweite Teil des Kapitels zeigt dann die optimierte Konfiguration der Datenbank.

**Kapitel 5** fasst die Ergebnisse zusammen und gibt einen Ausblick, welche weiteren Schritte für die Zukunft sinnvoll erscheinen.

**Anhang A** präsentiert die wichtigsten Konfigurationsparameter sowohl für den Datenbankmanager als auch für die Datenbank.

**Anhang B** beschreibt den Inhalt der beiliegenden CD.

## 1.4 Danksagungen

Die hier vorgelegte Diplomarbeit wird in der vorliegenden Form ohne die Hilfe und Unterstützung von Fachkollegen und Freunden nicht möglich gewesen. Mein besonderer Dank gilt daher:

- Karsten Tolle für seine kompetente Betreuung, den fachlichen Austausch und seine Unterstützung in schwierigen Momenten.
- Stephan Fröhder für das Durchlesen und Korrigieren des Manuskripts.

Nicht zuletzt danke ich meiner Frau Blandine für ihr Verständnis und ihre jahrelange Unterstützung und möchte ihr diese Arbeit widmen.



# Kapitel 2

## Grundlagen

In diesem Kapitel werden die grundlegenden Arbeitsmittel, die bei dieser Arbeit verwendet werden vorgestellt.

Der erste Abschnitt beschreibt eine Einführung in die Optimierung. Dann wird der Aufbau der Anwendung (RDF-S3) beschrieben und anschließend wird das Datenbankverwaltungssystem (IBM DB2 v8.1) vorgestellt.

### 2.1 Einführung in die Optimierung

Mit *Leistung* (*Performance*) wird die Art und Weise bezeichnet, wie ein Computersystem unter einer bestimmten Auslastung arbeitet [DB2-02]. Außerdem wird die Leistung an einem oder mehreren Faktoren des Systems wie Antwortzeit, Durchsatz und Verfügbarkeit gemessen. Die *Antwortzeit* ist die Ausführungsdauer von ausgeführten Operationen einer Anwendung. Mit *Durchsatz* wird die Anzahl der Transaktionen pro Sekunde gemessen. Die *Verfügbarkeit* bezeichnet die Zeit, in der das System ununterbrochen aktiv<sup>1</sup> ist. Die Leistung kann von verschiedenen Faktoren beeinflusst werden:

- Von den im System verfügbaren Ressourcen
- Von der Auslastung der Ressourcen und
- Vom Ausmaß der gemeinsamen Nutzung dieser Ressourcen

#### 2.1.1 Warum optimieren?

Im Allgemeinen optimiert man das System, wenn das Kosten-Nutzen-Verhältnis verbessert werden soll. Dabei können die folgenden speziellen Optimierungsziele verfolgt werden:

- Steigerung der Arbeitsbelastung ohne steigende Verarbeitungskosten. Zum Beispiel soll eine Steigerung der Auslastung erreicht werden, ohne dass neue Hardware erworben oder mehr Prozessorzeit in Kauf genommen werden muss.

---

<sup>1</sup>Erreichbar, verfügbar

- Minimierung der Systemantwortzeiten bzw. Erhöhung des Durchsatzes ohne Steigerung der Verarbeitungskosten.
- Reduzierung der Verarbeitungskosten ohne negative Auswirkungen für die Benutzer.

### *Was sollte bei der Leistungsoptimierung beachtet werden?*

Bei der Leistungsoptimierung sollten u.a. auch die folgenden Punkte beachtet werden:

**Das ganze System als Ganzes zu betrachten:** Ein Parameter bzw. System lässt sich nicht isoliert optimieren.

**Jeweils nur einen Parameter gleichzeitig ändern:** Man sollte zur Leistungsoptimierung nicht mehrere Parameter in einem Schritt ändern. Selbst wenn es sicher ist, dass alle Änderungen vorteilhaft sein werden, hat man hinterher keine Möglichkeit, den Beitrag jeder Änderung zu bewerten. Wenn aber jeweils nur ein Parameter geändert wird, so hat man einen Vergleichspunkt und kann feststellen, ob die Änderung die gewünschte Wirkung erzielt hat.

**Messungen und Neukonfigurierungen nach Ebenen durchführen:** Analog empfiehlt es sich, die einzelnen Ebenen des Systems getrennt voneinander zu optimieren. Die folgende Liste von Ebenen innerhalb eines Systems kann dabei als Richtlinie dienen:

- Hardware
- Betriebssystem
- Datenbankmanager
- Anwendungsprogramme
- SQL-Anweisungen
- Anwendungsserver und -requester

### **Entwicklung eines Prozesses der Leistungsverbesserung**

Der Leistungsoptimierungsprozess ist ein iteratives und langfristig angelegtes Verfahren zur Überwachung und Optimierung von Leistungsbereichen. Abhängig von den Überwachungsergebnissen sollte die Konfiguration des Datenbankservers angepasst werden und falls erforderlich Änderungen an den Anwendungen vorgenommen werden, die den Datenbankserver verwenden. Für den Prozess der Leistungsoptimierung kann man als Richtlinie die folgende Prozedur benutzen:

#### **Prozedur:**

1. Leistungsziele definieren.
2. Festlegen der Leistungsindikatoren für die wichtigsten Leistungsprobleme im System.

3. Entwickeln und ausführen eines Leistungsüberwachungsplanes.
4. Analysieren der Ergebnisse der Überwachung, um zu ermitteln, welche Ressourcen optimiert werden müssen.
5. Jeweils nur eine Anpassung pro Schritt.

Die Verbesserung der Leistung durch Optimieren des Datenbankservers und der Anwendungen lässt sich nur bis zu bestimmten Punkt vorantreiben. Wenn dieser Punkt erreicht ist, bleibt nur die Möglichkeit, die Hardware aufzurüsten.

### Was ist hilfreich bei der Leistungsoptimierung?

DB2<sup>®</sup> verfügt über verschiedene integrierte Assistenten, auf die man bei der Durchführung einiger leistungsrelevanter Verwaltungsaufgaben zurückgreifen kann. Deshalb sollten, beim Starten eines neuen Exemplars von DB2, die folgenden Empfehlungen für eine Ausgangskonfiguration in Betracht gezogen werden:

- Konfigurationsadvisor in der Steuerzentrale verwenden, um Empfehlungen für sinnvolle Standardwerte für das System zu erhalten. Die Standardwerte, die für DB2 voreingestellt sind, sollten für die vorhandene Hardwareumgebung und eigene Bedürfnisse optimiert werden.
- Weitere Assistenten in der Steuerzentrale verwenden. Solche Aufgaben sind in der Regel diejenigen, bei denen bedeutende Leistungsverbesserungen mit geringem Zeitaufwand und wenig Mühe erreicht werden. Solche Assistenten sind: Datenbank erstellen, Tabelle erstellen, Index erstellen und Aktualisierung auf mehreren Systemen konfigurieren. Der Auslastungsleistungsassistent (Designadvisor) führt durch die Aufgaben, die bei der Optimierung von Abfragen helfen. Die Diagnosezentrale stellt einige Überwachungs- und Optimierungstools bereit.
- Die Übersichtstabellen<sup>2</sup> konsultieren, die jeden verfügbaren Konfigurationsparameter für den Datenbankmanager und für jede Datenbank auflisten und kurz beschreiben. Auf diesen Übersichtstabellen ist eine Spalte enthalten, die angibt, ob eine Änderung des jeweiligen Parameters einen hohen, mittleren, niedrigen oder keinen Einfluss auf die Leistung im positiven oder negativen Sinn hat.

## 2.2 RDF-Source related Storage System

Die Anwendung *RDF-Source related Storage System* (RDF-S3), die Gegenstand dieser Arbeit ist, ist vollständig in Java programmiert und somit plattformunabhängig, was nicht zu vernachlässigen ist. Die Hauptaufgabe dieser Anwendung ist das Speichern von RDF-Daten (RDF-Tripeln). Darüberhinaus werden in RDF-S3 zusätzliche Informationen über

---

<sup>2</sup>Siehe Anhang A

Quellen (Quellinformationen) für jedes Tripel gespeichert. Dadurch wird die Rückverfolgung ermöglicht, d.h. die Benutzer können nachvollziehen, woher die Informationen stammen oder von wem und wann sie geändert worden sind. Nur so wiederum kann z.B. die Glaubwürdigkeit und Aktualität der Informationen überprüft werden. Das ist sehr wichtig, denn sonst könnten die Benutzer den Informationen nicht trauen, da jede Person beliebige Aussagen hinzufügen kann. Zusätzlich kann man bei der Quelle nach weiteren relevanten Informationen suchen.

Die Zusatzinformationen können auf verschiedene Weise gespeichert werden. Eine Möglichkeit ist, die vorhandenen Tripel zu Quadrupel zu erweitern, wobei die Zusatzinformationen als vierter Teil innerhalb von RDF-Aussagen gespeichert werden. In vielen Ansätzen wird dabei der vierte Teil innerhalb der Tripel verwendet. Dies ist jedoch nicht konform mit dem RDF-Model. Viele existierende RDF-Anwendungen könnten hiermit Schwierigkeiten haben. Die Lösung dieses Problems wäre, die zusätzlichen Informationen, in einer Art *eingeschränkter Quadrupel* zu speichern, der vierte Teil würde nicht innerhalb der Tripel verwendet. Alle existierenden RDF Anwendungen könnten weiterhin verwendet werden, indem sie die Zusatzinformationen ignorieren und es muss auf die Zusatzinformationen nicht verzichtet werden [T04].

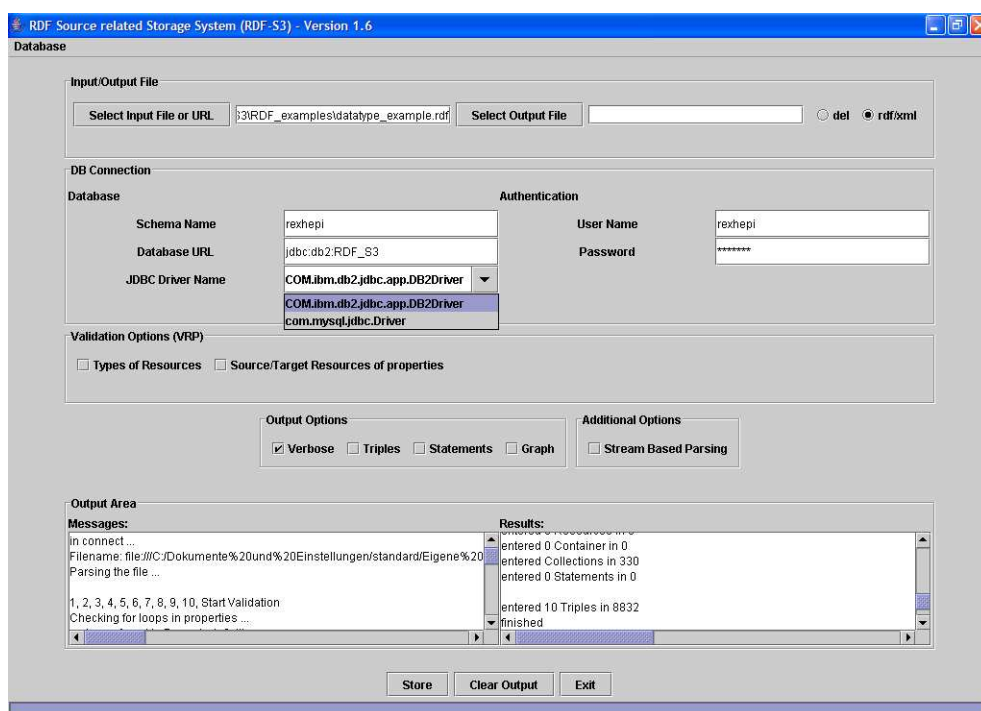


Abbildung 2.1: Screenshot von RDF-S3.

RDF-S3 verwendet diesen Ansatz und speichert zu jeder Aussage die zusätzlichen Informationen — die Quellinformationen. RDF-S3 ist eine Anwendung, die dem Benutzer

viele Vorteile bringt:

- Die Rückverfolgung der Quellen von Informationen, um dort nach weiteren Informationen zu suchen.
- Die Option, anhand des Speicherdatums die Aktualität der Informationen zu überprüfen.
- Durch minimale Änderungen kann das Speichern von verschiedenen Versionen von Dokumenten nebeneinander gemacht werden, sodass die Entwicklung dokumentiert wird.
- Das Löschen oder Updaten von gespeicherten Quellen.
- RDF-S3 kann mit jedem SQL konformen RDBMS<sup>3</sup> benutzt werden.
- Es bietet grafische Benutzeroberfläche (GUI) für die einfache Benutzung.
- Es bietet eine API (Application Program Interface) für die Behandlung der zusätzlichen Informationen.

Erwähnenswert ist auch, dass RDF-S3 durch die Kombination der beiden gängigen Speicherstrategien: Generische Repräsentation<sup>4</sup> und Schemaspezifische Repräsentation<sup>5</sup>, die spezifischen Nachteile der einzelnen Methoden umgehen kann.

Die Abbildung 2.2 zeigt die innere Struktur von RDF-S3. Es besteht aus dem so ge-

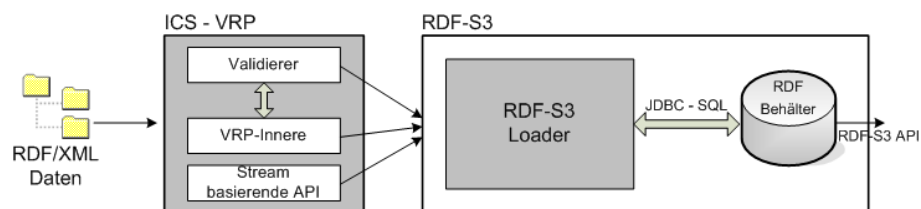


Abbildung 2.2: Die Arbeitsweise von RDF-S3.

nannten Loader, der die Daten in einem Behälter speichert und einer API, die den Zugriff auf RDF-Daten zusammen mit Quellinformationen ermöglicht. Der Loader interagiert mit dem ICS-Validating RDF Parser (VRP) [T00], der dem Benutzer die Möglichkeit bietet die RDF-Daten, bevor sie gespeichert werden, auf der semantischen Ebene auf Gültigkeit zu überprüfen. Dies benötigt ein inneres Speicher-Modell. Für sehr große RDF-Dateien kann das ein Nachteil sein, denn die Speichergröße ist begrenzt. Deswegen ermöglicht RDF-S3

<sup>3</sup>Relational Database Management System

<sup>4</sup>Generischer Ansatz basiert auf der strukturellen Abstraktion des Datenmodells und ist unabhängig von konkreten Schemata.

<sup>5</sup>Schemaspezifischer Ansatz basiert auf der semantischen Abstraktion des Datenmodells und ist abhängig von einem konkreten Schemata.

zusätzlich auch die *Stream Based Parsing*-Methode zum Speichern der Daten, d.h. es wird kein inneres Speicher-Modell von VRP aufgebaut. Allerdings, dies kann länger dauern, da RDF-S3 alle Tripel einzeln einfügen muss und kann nicht auf Informationen von VRP, die bei der Validierung gewonnen werden, zugreifen.

Bevor die Daten gespeichert werden können, muss die Datenbank zuerst initialisiert werden. Dabei wird eine Reihe von Tabellen erstellt, wie z.B.:

- *Resources* – Enthält alle Resources und Literale<sup>6</sup>.
- *Namespaces* – Enthält Abkürzungen von URIs von Klassen, Eigenschaften usw.
- *Sources* – Enthält Informationen über bereits eingefügten Quellen.
- *Literals* – Enthält Literale, den Typ und die Sprachspezifikationen.
- *Classes* – Enthält alle gespeicherten Klassen.
- *Properties* – Enthält alle gespeicherten Eigenschaften.
- *Containers* – Enthält alle gespeicherten Container.

Zusätzlich es werden auch *Classes\_Source\_Usage*, *Properties\_Source\_Usage* und *Containers\_Source\_Usage* erstellt, die jeweils Informationen über Beziehungen von Quellen enthalten. Außerdem es werden noch die folgenden Tabellen erstellt: POI, Subclasses und Subproperties. Weitere Informationen sind in [TW04] zu finden.

Damit beide Speicherstrategien ihre jeweiligen Vorteile nutzen können, werden in RDF-S3 die Daten redundant gespeichert. Durch die Verschlüsselung der URIs mit Zahlen wird hierbei zusätzlicher Speicherverbrauch reduziert. Außerdem, die Redundanz der Daten bringt auch das Problem der Synchronisation der Daten mit sich. Um dieses Problem zu lösen, werden in RDF-S3 Fremdschlüssel benutzt. Da aber die Synchronisation der Daten nur beim Löschen oder Einfügen geschieht, wird die Performance von Abfragen nicht beeinflusst.

## 2.3 Das Datenbanksystem

In diesem Abschnitt wird das Datenbanksystem IBM DB2 v8.1 und seine wichtigsten Komponenten kurz beschrieben.

### 2.3.1 Architektur und Prozesse

Kenntnisse über Architektur und Prozesse für DB2 UDB<sup>TM</sup> (UDB – Universal Database) sind wichtig bei der Bestimmung der Werte für viele Konfigurationsparameter (z.B. Anzahl asynchroner Seitenlöschfunktionen (*num\_iocleaners*), max. Anzahl der Agenten (*maxagents*) die gleichzeitig tätig sein können usw.), die wiederum sehr wichtig für die Optimierung sind.

---

<sup>6</sup>Literale sind lesbare Texte und Textfragmente, Zahlen, Datumsangaben usw.

## Übersicht über die Architektur und Prozesse von DB2

Die Abbildung 2.3 auf Seite 19 zeigt eine allgemeine Übersicht über die Architektur und die Prozesse für DB2 UDB.

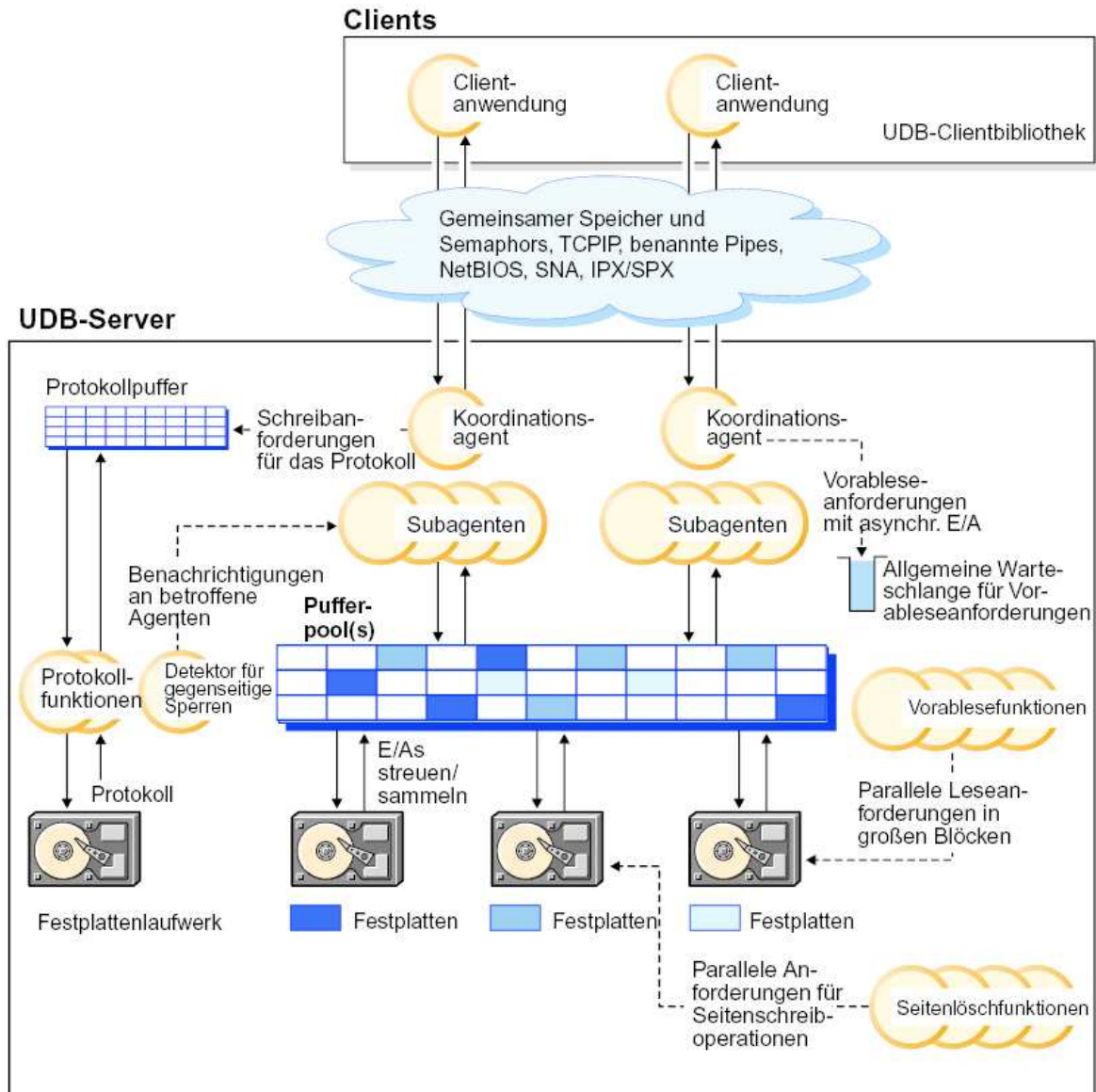


Abbildung 2.3: Architektur- und Prozessübersicht [DB2-02]

Auf der Clientseite gibt es lokale und/oder ferne Anwendungen, die mit der Clientbibliothek von DB2 Universal Database verbunden sind. Für die Kommunikation benutzen lokale Clients einen gemeinsamen Speicher und Semaphoren. Die fernen Clients verwenden unter anderem auch ein Protokoll wie benannte Pipes (Named Pipes – NPIPE) oder

TCP/IP usw. (Abb. 2.3).

Auf der Serverseite werden die Aktivitäten durch so genannte EDUs (Engine Dispatchable Units - zuteilbare Einheiten der Steuerkomponente) gesteuert. In der Abbildung sind die EDUs als Kreise oder Gruppen von Kreisen dargestellt. Die am weitesten verbreitete Art von EDUs sind die DB2-Agenten, die den größten Teil der SQL-Verarbeitung im Auftrag von Anwendungen ausführen. Weitere häufig genutzte Arten von EDUs sind Vorablesefunktionen (Prefetchers) und Seitenlöschfunktionen (Page cleaners).

Mithilfe eines Poolfunktionsalgorithmus, der alle Agenten und Subagenten verwaltet, wird die Häufigkeit der Erstellung und Vernichtung von EDUs so gering wie möglich gehalten.

**Pufferpools** (Buffer pool) sind Bereiche des DatenbankserverSpeichers, in die Datenbankseiten mit Benutzertabellendaten, Indexdaten und Katalogdaten temporär eingelesen und dort modifiziert werden können. Pufferpools spielen eine sehr wichtige Rolle, wenn es um die Datenbankleistung geht, weil der Zugriff auf Daten im Hauptspeicher wesentlich schneller als auf Daten auf dem Plattenspeicher erfolgen kann. Wenn größere Teile der von Anwendungen benötigten Daten, in einem Pufferpool vorhanden sind, ist für den Zugriff auf die Daten weniger Zeit erforderlich, als wenn sie z.B. auf einer Festplatte gesucht werden müssen.

Die Konfiguration der Pufferpools sowie der EDUs für Vorablesefunktionen und Seitenlöschfunktionen bestimmt, wie schnell auf Daten zugegriffen werden kann und wie schnell sie Anwendungen verfügbar gemacht werden können.

- **Vorablesefunktionen** rufen Daten von der Festplatte ab und lesen sie in den Pufferpool ein, bevor sie von Anwendungen benötigt werden. Wenn es zum Beispiel keine Vorablesefunktionen für Daten gäbe, müssten Anwendungen darauf warten, dass Daten vom Plattenspeicher in den Pufferpool gelesen werden und könnten erst dann mit den Daten arbeiten.
- **Seitenlöschfunktionen** speichern Daten aus dem Pufferpool zurück auf den Plattenspeicher. Seitenlöschfunktionen sind im Hintergrund aktive EDUs, die von den Anwendungsagenten unabhängig ihre Arbeit verrichten. Die nicht länger benötigten Seiten im Pufferpool werden von Seitenlöschfunktionen auf den Plattenspeicher zurückgeschrieben. So wird sicher gestellt, dass im Pufferpool Platz für Seiten ist, die von Vorablesefunktionen eingelesen werden.

Weitere wichtige EDUs sind:

- *E/A-Server* (I/O-Server).– Zur Aktivierung des Vorablesezugriffs startet der Datenbankmanager zum Lesen von Datenseiten separate Steuerthreads, die als E/A-Server bezeichnet werden. Wenn eine ausreichende Anzahl von E/A-Servern (*num\_ioservers*) vorhanden ist, kann die Leistung von Abfragen erheblich gesteigert werden. Es empfiehlt sich, die Anzahl der E/A-Server der Anzahl der physischen Platten anzupassen, um die Möglichkeit für parallele E/A-Operationen zu maximieren. Es ist sogar besser, die Anzahl von E/A-Servern zu hoch als zu niedrig anzusetzen. Die zusätzlichen

E/A-Server werden einfach nicht verwendet und infolgedessen wird die Leistung nicht beeinträchtigt.

- *Detektor für gegenseitiges Sperren* (Deadlock detector).– Wenn mehrere Anwendungen mit Daten aus der Datenbank arbeiten, kann es zwischen zwei oder mehreren von ihnen zu gegenseitigen Sperren kommen. Gegenseitige Sperren entstehen, wenn eine Anwendung (z.B. A1) darauf wartet, dass eine andere Anwendung (z.B. A2) eine Sperre für Daten freigibt und gleichzeitig die andere Anwendung (A2) auf die Freigabe von der ersten Anwendung (A1) wartet. Gegenseitiges Sperren kann die Performance beeinträchtigen. Der Detektor für gegenseitiges Sperren wählt eine beliebige der gegenseitig gesperrten Anwendungen aus und gibt die Sperren frei.
- *Protokollfunktionen*.– Alle Datenbanken pflegen Protokolldateien, die Datensätze über Datenbankänderungen aufzeichnen d.h. Änderungen an regulären Daten- und Indexseiten. Die Änderungen werden zuerst im Protokollpuffer und anschließend vom Protokollprozess auf die Festplatte geschrieben. Das Protokollieren dient zur einfachen Wiederherstellung der Daten z.B. nach einem Ausfall des Datenbanksystems.

### Der Ablauf einer Aktion im Client-/Servermodell

Um die vorgestellten Komponenten des Systems zu einer Gesamtbild zusammenzuführen, wird der Ablauf einer Aktion beschrieben.

Sowohl lokale als auch ferne Anwendungsprozesse können in DB2 mit derselben Datenbank arbeiten. Bevor der Datenbankmanager die Aufgabe einer Anwendung erledigen kann, wird mithilfe eines EDUs eine Kommunikationsverbindung zwischen den beiden eingerichtet. Die fernen Clients müssen zusätzlich eine TCP/IP Verbindung aufbauen. Dann wird vom DB2 ein Koordinationsagent für die Anwendungsanforderungen zugewiesen, der Subagenten für die Ausführung einzelner Aktionen des Client beauftragt. Der Subagent übersetzt die Anfrage und leitet die Anfrage an den Pufferpool-Manager weiter, um zu überprüfen, ob die gesuchten Daten in dem von ihm verwalteten Speicher vorhanden sind. Ist das nicht der Fall, werden die gesuchten Daten von der Festplatte in den Hauptspeicher transferiert, was eine Aufgabe des E/A-Servers ist. Gleichzeitig überprüfen die Vorabtestfunktionen, ob die Möglichkeit besteht, dass noch mehr Daten benötigt werden, die durch die E/A-Server ebenfalls in den Pufferpool transportiert werden. Wenn die gesuchten Daten im Pufferpool sind, werden sie vom zuständigen Subagenten an den Koordinationsagenten geliefert, der entweder die Daten weiter bearbeitet, oder sie direkt an den Client sendet. Parallel dazu werden alle Datenbankänderungen vom Protokoll-Agenten protokolliert und können bei der Wiederherstellung (RESTORE) benutzt werden.

### 2.3.2 Übersicht über Speicherobjekte

Mithilfe der folgenden Datenobjekte kann definiert werden, wie Daten auf dem System gespeichert werden und wie die Leistung verbessert werden kann:

- Behälter
- Tabellenbereiche
- Pufferpools<sup>7</sup>

## Behälter

Ein *Behälter* (Container) ist eine physische Speichereinheit. Er kann als ein Verzeichnis (Directory), eine unformatierte Einheit (Raw Device) oder eine Datei (File) fungieren.

Ein Behälter ist einem einzigen Tabellenbereich zugeordnet, ein einzelner Tabellenbereich hingegen kann sich über viele Behälter erstrecken. Abbildung 2.4 zeigt die Beziehung zwischen Tabellenbereichen, Tabellen und Behältern.

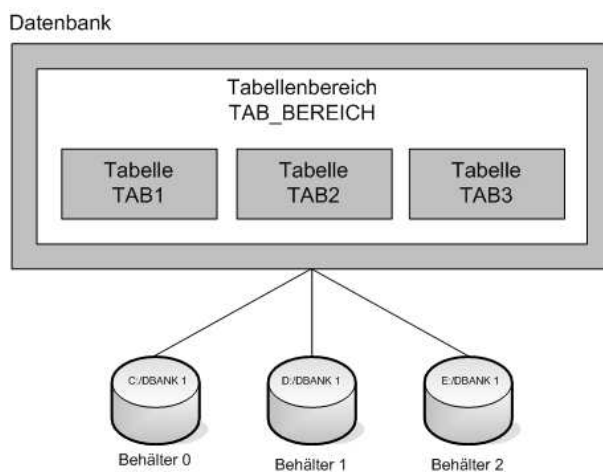


Abbildung 2.4: Tabellenbereiche und Tabellen in einer Datenbank.

Der Tabellenbereich TAB\_BEREICH, der sich über die Behälter 0, 1 und 2 erstreckt, enthält alle drei Tabellen: TAB1, TAB2 und TAB3. Dieses Beispiel zeigt jeden Behälter auf einer getrennten Platte.

Daten aller Tabellen werden in allen Behältern reihum verteilt gespeichert. Dies sorgt für eine gleichmäßige Verteilung der Daten auf die Behälter, die zu einem bestimmten Tabellenbereich gehören. Die Anzahl von Seiten, die der Datenbankmanager in einen Behälter schreibt, bevor er zu einem anderen Behälter wechselt, wird als Speicherbereich (*Extent*) bezeichnet.

## Tabellenbereiche

Eine Datenbank wird in Bestandteilen verwaltet, die als *Tabellenbereiche* (Tablespaces) bezeichnet werden. Ein Tabellenbereich ist ein Platz zum Speichern von Tabellen. Eine Tabelle besteht aus Daten wie z.B. Benutzerdaten, Systemdaten, Indizes usw., die logisch

<sup>7</sup>Bereits erläutert.

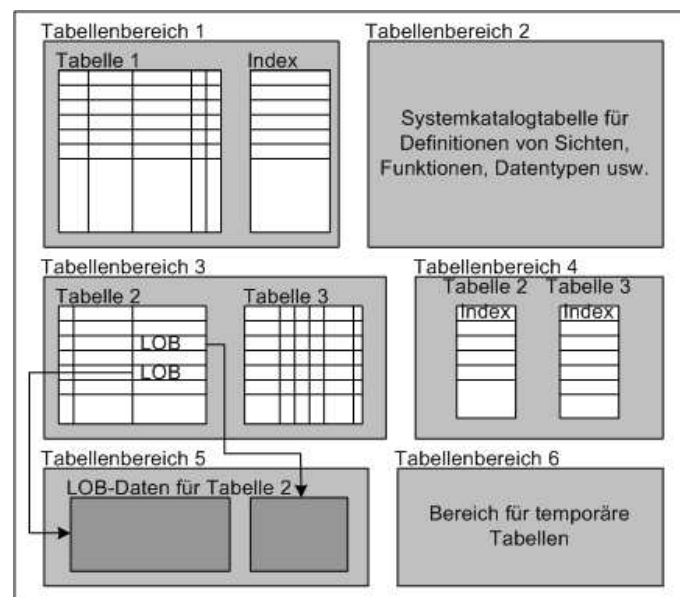


Abbildung 2.5: Übersicht über Tabellenbereiche.

in Spalten und Zeilen angeordnet sind. Außerdem können die Tabellenbereiche über mehr als nur eine physische Speichereinheit (Behälter) verteilt werden. Die Abb. 2.5 auf Seite 23 verdeutlicht das Konzept der Verteilung von Daten auf verschiedene Tabellenbereiche.

In DB2 werden zwei Arten von Tabellenbereichen unterstützt: vom System verwaltete Tabellenbereiche (SMS – System Managed Space) und vom Datenbankmanager verwaltete Tabellenbereiche (DMS – Database Managed Space) (Abb. 2.6 auf Seite 24). SMS–Tabellenbereiche sind eine sehr gute Wahl für allgemeine Zwecke. Sie bieten eine gute Leistung bei geringem Verwaltungsaufwand. Wenn aber mit etwas mehr Verwaltungsaufwand eine bessere Leistung erzielt werden soll, sind die DMS–Tabellen die richtige Wahl. DMS–Tabellenbereiche in unformatierten Einheitenbehältern bieten die beste Leistung, weil keine doppelte Pufferung erfolgt. In DMS–Tabellenbereiche in Dateibehälter und SMS–Tabellenbereiche hingegen kann es zu einer doppelten Pufferung der Daten kommen d.h. die Daten werden zuerst auf der Datenbankmanagerebene und anschließend noch einmal auf der Dateisystemebene in Puffern zwischengespeichert, was einen zusätzlichen Aufwand bedeutet.

- **SMS–Tabellenbereiche** speichern Daten in Betriebssystemdateien d.h. jeder Behälter ist ein Verzeichnis im Dateibereich des Betriebssystems. Die Daten in den Tabellenbereichen werden Einheiten übergreifend in Speicherbereichen (eine Gruppe von aufeinander folgenden Seiten) in allen Behältern des Systems gespeichert. Um eine gleichmäßige Verteilung der Speicherbelegung auf alle Behälter im Tabellenbereich zu garantieren, werden die Anfangsspeicherbereiche für Tabellen reihum in allen Behältern angelegt. Eine solche Verteilung der Speicherbereiche ist von großer Bedeutung, wenn es eine große Zahl kleiner Tabellen gibt.

- **DMS-Tabellenbereiche.**– Bei einem DMS-Tabellenbereich (vom Datenbankmanager verwalteter Tabellenbereich) steuert der Datenbankmanager den Speicherbereich. Jeder Behälter ist entweder eine im Voraus zugeordnete Datei fester Größe oder eine physische Einheit wie eine Platte. Um eine gleichmäßige Verteilung von Daten auf alle Behälter sicherzustellen, verwenden die DMS-Tabellenbereiche, ebenso wie SMS-Tabellenbereiche und SMS-Behälter, Einheiten übergreifendes Lesen und Schreiben von Daten (Striping).

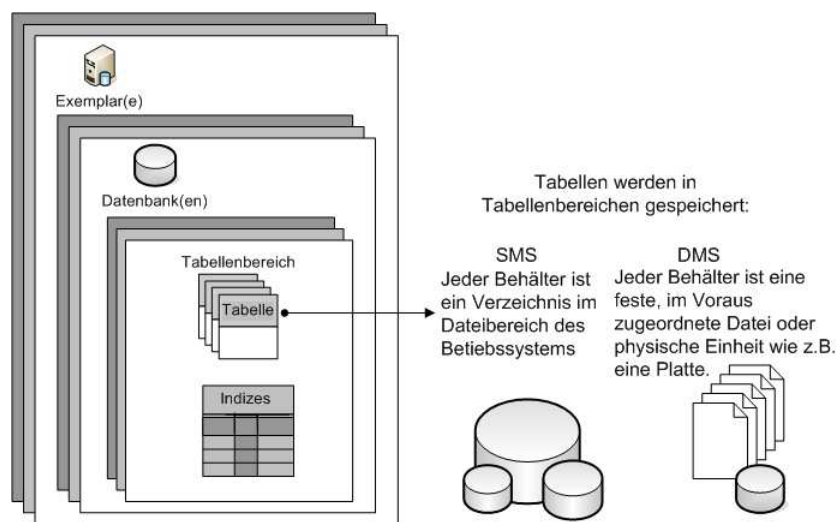


Abbildung 2.6: SMS- und DMS-Tabellenbereiche.

DMS-Tabellenbereiche unterscheiden sich von SMS-Tabellenbereichen dadurch, dass bei DMS-Tabellenbereichen der Speicherbereich bereits bei der Erstellung des Tabellenbereichs zugeordnet wird und nicht erst, wenn er benötigt wird.

Darüber hinaus kann sich die Platzierung von Daten bei beiden Arten von Tabellenbereichen unterscheiden. Bei effizienten Tabellensuchoperationen ist es z.B. wichtig, dass die Seiten in einem Speicherbereich physisch aufeinander folgen. Bei SMS-Tabellenbereichen entscheidet das Dateisystem des Betriebssystems, wo die logischen Dateiseiten physisch abgelegt werden. Je nach Umfang anderer Aktivitäten im System, kann das Dateisystem die Datenseiten aufeinander folgend zuordnen, muss dies aber nicht. Bei DMS-Tabellenbereichen jedoch kann der Datenbankmanager sicherstellen, dass die Seiten physisch aufeinander folgen<sup>8</sup>, da er direkt mit dem Plattendatenträger interagiert d.h. er ist unabhängig vom Dateisystem des Betriebssystems. Dies ermöglicht auch einen höheren E/A-Durchsatz. Dennoch sind für temporäre Tabellenbereiche SMS-Bereiche den DMS-Bereichen vorzuziehen:

1. Die Erstellung einer temporären Tabelle in DMS-Tabellenbereichen erfordert mehr Systemaufwand.

<sup>8</sup>Nur bei unformatierten Einheiten, während dies bei Dateibehältern nicht immer der Fall ist.

2. Plattenspeicherplatz wird im SMS nach Bedarf zugeordnet, wohingegen er im DMS zuvor zugeordnet werden muss. Eine vorherige Zuordnung kann ein Problem darstellen: Temporäre Tabellenbereiche enthalten Übergangsdaten, die variablen Speicherbedarf haben können, der durch die vorherige Zuordnung verloren wäre.
3. Da der Datenbankmanager auch die temporären Tabellenseiten im Speicher zu behalten versucht, um sie nicht auf die Platte auszulagern, sind die Leistungsvorteile von DMS weniger signifikant.

### 2.3.3 Tabellen und Indizes

Das Verständnis der Organisation von Tabellen und Indizes kann bei der Optimierung der Verwendung dieser Objekte helfen. Die Tabelle besteht aus Daten, die logisch in Zeilen und Spalten angeordnet sind d.h. eine Liste aus Datensätzen (Records) darstellen.

#### Tabellen

DB2 stellt zwei Arten von Tabellen bereit:

- Standardtabellen und
- MDC-Tabellen (Multi-Dimensional Clustering-Tabellen)

**Standardtabellen** sind die Tabellen, in denen die Zeile die höchste logische Struktur unter der Tabelle ist<sup>9</sup>. Tabellendaten sind logisch in Form einer Liste von Datensätzen organisiert. Diese Datensätze werden logisch in Abhängigkeit von der Größe des Tabellenbereichs (*dft\_extent\_sz* – Parameter) dann zu Gruppen zusammengefasst. Wenn z.B. die Größe des Tabellenbereichs drei ist, sind die Seiten Null bis Zwei Teil des ersten Speicherbereichs, die Seiten Drei bis Fünf sind Teil des zweiten usw. Die Größe der Datensätze und die Größe der Datensätze bestimmt die Anzahl der Datensätze, die eine einzelne Datensatzseite aufnehmen kann. Die meisten Seiten enthalten nur Benutzerdatensätze.

Zur Verwaltung der Tabellen verwendet DB2 spezielle interne Datensätze, die nur eine geringe Anzahl von Seiten benötigen. Auf jeder 500sten Seite befindet sich beispielsweise ein FSCR-Datensatz (Free Space Control Record, Steuersatz für freien Speicherbereich). Diese Datensätze enthalten Informationen über den vorhandenen freien Speicherbereich bis zum nächsten FSCR-Datensatz und werden verwendet, wenn Datensätze in die Tabelle eingefügt werden. Die Einstellung der Registriervariablen DB2MAXFSCRSEARCH bestimmt z.B. die Anzahl von FSCRs in einer Tabelle, die für eine INSERT-Anweisung durchsucht werden. Wenn eine höhere INSERT-Geschwindigkeit erwünscht ist, dann ist ein kleiner Wert für DB2MAXFSCRSEARCH von Vorteil, allerdings mit der möglichen Folge einer schlechteren Speicherplatznutzung.

**MDC-Tabellen (Multi-Dimensional Clustering-Tabellen)** sind die Tabellen, in denen die höchste logische Struktur unterhalb der Tabelle der Block ist<sup>10</sup>. Die MDC-Tabellen

---

<sup>9</sup>Siehe Abbildung 2.7 auf Seite 26.

<sup>10</sup>Siehe Abbildung 2.8 auf Seite 27.

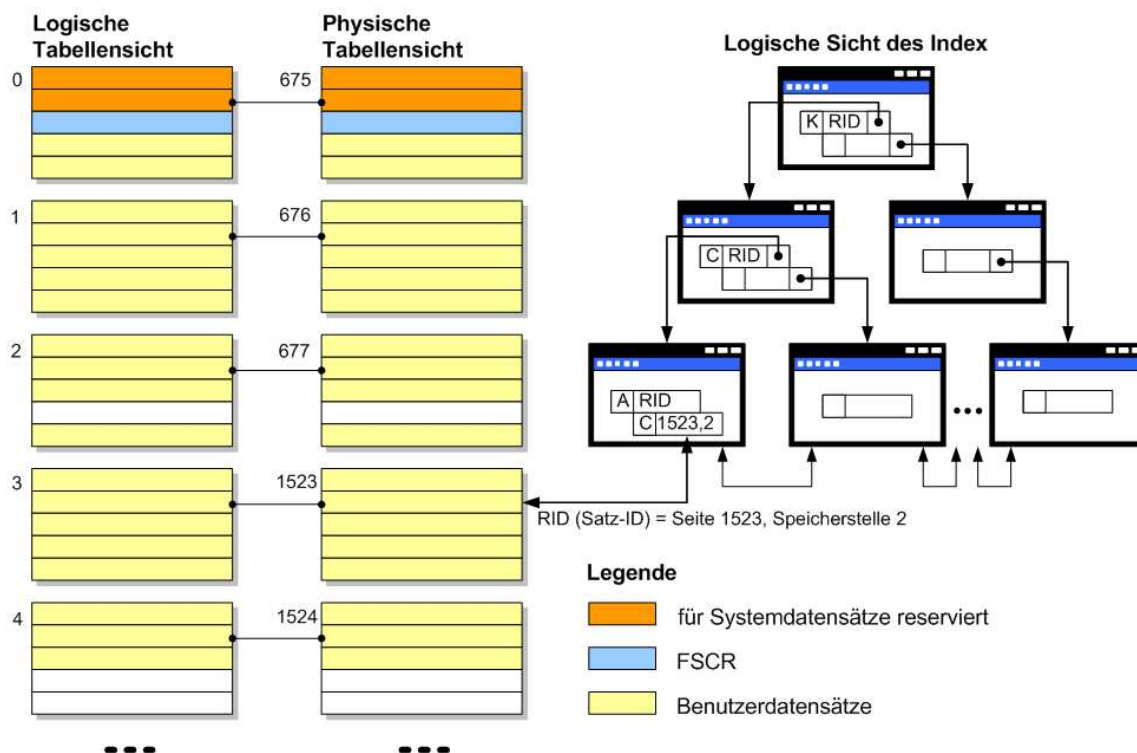


Abbildung 2.7: Logische Tabellen-, Datensatz und Indexstruktur für Standardtabellen.

sind logisch gesehen wie die Standardtabellen organisiert d.h. in Form einer Liste in Datensätzen. Darüber hinaus sind die Seiten der MDC-Tabellen in Blöcken gruppiert. Wie aus den Namen dieser Tabellen zu entnehmen ist, ordnen die MDC-Tabellen ihre Daten in mehr als nur einer Dimension, in so genannten Clustern (Datengruppen) an. Jede Dimension kann durch eine oder mehrere Spalten bestimmt werden. Z.B. durch das Definieren einer Dimension in einer Spalte JAHR und einer anderen Dimension in einer Spalte MONAT werden alle Datenzeilen desselben Jahres und desselben Monats auf dem Festplattenlaufwerk zusammen angeordnet. Dies kann zu einer Steigerung der Leistung bei Abfragen führen, die Vergleichselemente einschließlich JAHR und/oder MONAT enthalten. Dimensionen können nicht mehr geändert werden, nachdem eine Tabelle erstellt ist.

Im Unterschied zu Standardtabellen bei MDC-Tabellen ist der erste Block komplett für Systemdatensätze reserviert. Die FSCR-Datensätze befinden sich am Anfang jedes Blocks und enthalten Informationen über den verfügbaren freien Speicher innerhalb des Blocks und nicht, wie bei Standardtabellen, über die folgenden 500 Seiten.

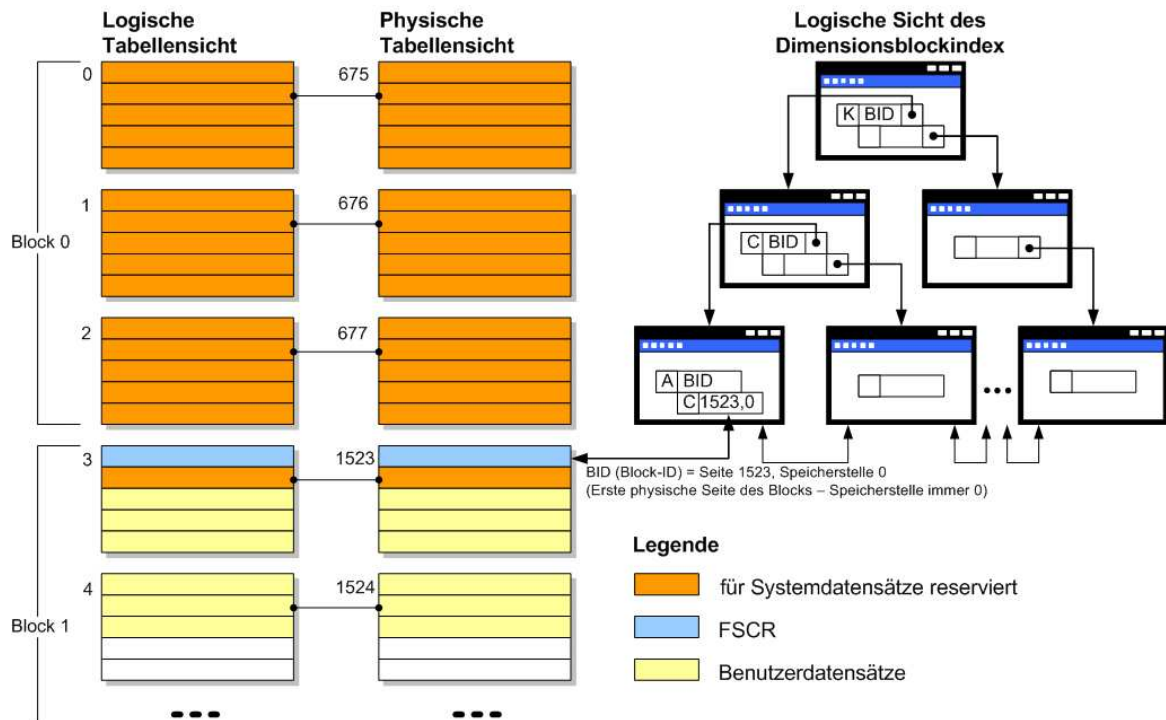


Abbildung 2.8: Logische Tabellen-, Datensatz- und Indexstruktur für MDC-Tabellen.

## Indexverwaltung

Um schnellen Zugriff auf die Datensätze in den Tabellendaten, die über einen Schlüsselwert<sup>11</sup> verfügen, zu ermöglichen, werden die Indexdaten als B-Baumstruktur organisiert. Die DB2-Indizes verwenden eine optimierte B-Baumstrukturimplementierung. Diese Implementierung ermöglicht einen hohen Grad des gemeinsamen Zugriffs und benutzt im Voraus schreibende Protokollierung (Write Ahead Logging). Während bei den Indizes für Standardtabellen die B-Baumstrukturimplementierung über bidirektionale Zeiger auf den Blattseiten verfügt, um in beide Richtungen (vorwärts und rückwärts) gerichtete Suchoperationen zu ermöglichen, muss bei den Indizes für MDC-Tabellen die Klausel `ALLOW REVERSE SCANS` angegeben werden. Für MDC-Tabellen werden automatisch ein Dimensionsblockindex, der Zeiger auf jeden Block für eine einzelne Clusterdimension enthält und ein zusätzlicher zusammengesetzter Blockindex erstellt, der alle Dimensionsschlüsselspalten enthält. Solche Indizes enthalten Zeiger auf Blöcke<sup>12</sup> bzw. Block-IDs (BID – Block ID) anstatt auf Datensätze bzw. Satz-IDs (RID – Record ID) und bieten Verbesserungen beim Datenzugriff. Der zusammengesetzte Blockindex dient zur Erhaltung der Clusterbildung bei `INSERT`- und `UPDATE`-Operationen.

<sup>11</sup>Für die eindeutige Identifikation der Datensätze, wird jedem Datensatz ein eindeutiger Schlüssel zugewiesen.

<sup>12</sup>Siehe Abbildung 2.8.

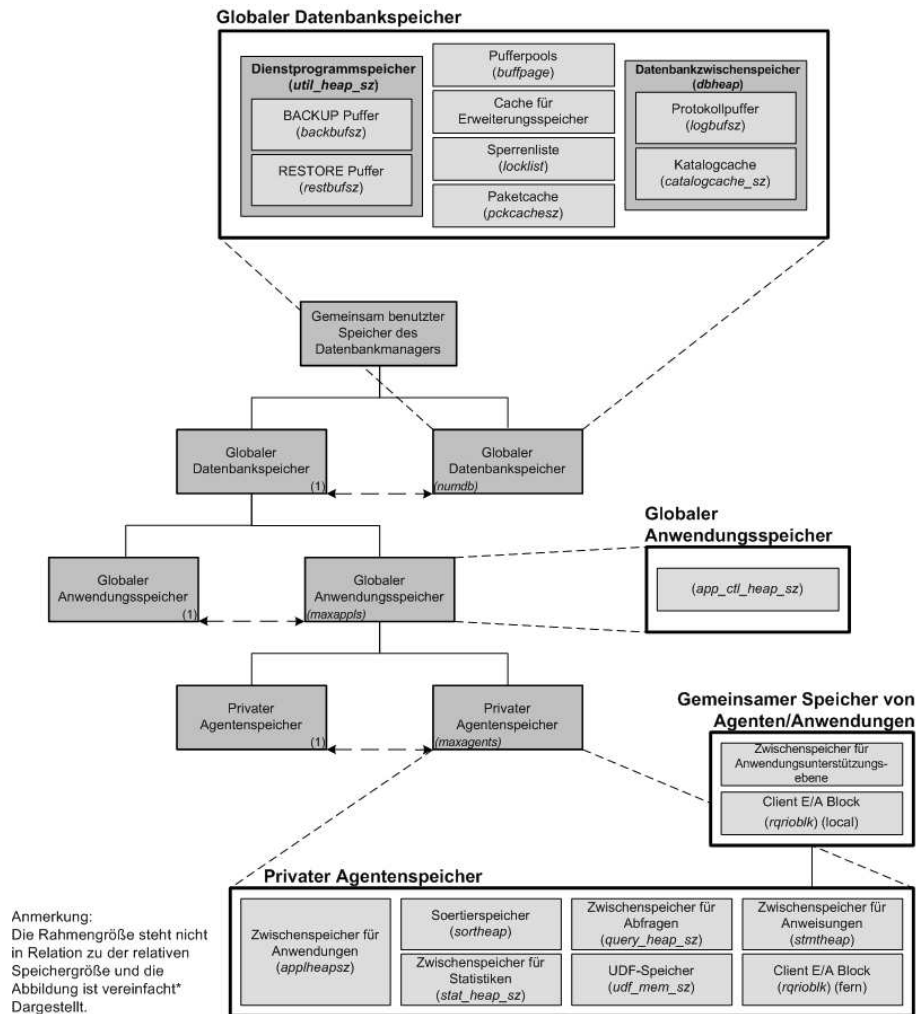


Abbildung 2.9: Arten und Verwendung des Speichers durch den Datenbankmanager.

### 2.3.4 Speicherverwaltung

Ein weiterer wichtiger Punkt bei der Leistungsoptimierung ist die Speicherverwaltung d.h. die Art und Weise, wie der verfügbare Speicher unter den Bereichen innerhalb der Datenbank zu verteilen ist. Diese Verteilung des Speichers wird in erster Linie durch viele Konfigurationsparameter abgestimmt. DB2 benutzt verschiedene Arten von Speichern wie: gemeinsamen Speicher des Datenbankmanagers, globalen Datenbankspeicher, globalen Anwendungsspeicher und privaten Agentenspeicher (Abb. 2.9).

Alle EDUs (Engine Dispatchable Unit) in einer Partition<sup>13</sup> sind mit dem gemeinsamen

<sup>13</sup>Eine Datenbankpartition ist ein Teil einer Datenbank, der aus seinen eigenen Daten, Indizes, Konfigurationsdateien und Transaktionsprotokollen besteht. Eine Datenbankpartition wird manchmal auch als Knoten oder Datenbankknoten bezeichnet.

Speicher des Exemplars (auch als gemeinsam benutzter Speicher des Datenbankmanagers bezeichnet) verbunden. Alle in einer Datenbank aktiven EDUs sind mit dem gemeinsamen Speicher dieser Datenbank verbunden. Alle aktiven EDUs, die für eine Anwendung zuständig sind, sind mit dem gemeinsamen Speicher dieser Anwendung verbunden. Schließlich verfügen alle EDUs auch über eigenen privaten Speicher (Abb. 2.9).

Gemeinsam benutzter Speicher des Datenbankmanagers wird z.B. beim Starten des Datenbankmanagers (`db2start`-Befehl) zugeordnet oder wenn eine Datenbank aktiviert wird oder wenn zum ersten Mal eine Verbindung zu ihr hergestellt wird. Alle anderen Arten von Speichern werden aus dem gemeinsamen Speicher des Datenbankmanagers verbunden oder zugeordnet. Durch den Konfigurationsparameter *instance\_memory* kann der gemeinsame Exemplarspeicher gesteuert werden.

Der gemeinsame Datenbankspeicher wird zugeordnet, wenn eine Datenbank aktiviert wird oder zum ersten Mal eine Verbindung zu ihr hergestellt wird. Für alle Anwendungen die eine Verbindung zur dieser Datenbank herstellen, wird dieser Speicher verwendet. Durch den Konfigurationsparameter *database\_memory* kann der gemeinsame Datenbankspeicher gesteuert werden.

Sowohl *instance\_memory* als auch *database\_memory* sind standardmäßig auf *automatic* gesetzt, sodass DB2 die Größe des dem Exemplar bzw. der Datenbank zugeordneten Speichers berechnet.

Aus dem Datenbankspeicher werden vielen Diensten Speicherbereiche zugeordnet. Dazu gehören unter anderem (Abb. 2.9):

- Pufferpools
- Sperrenliste (Locklist)
- Paketcache
- Datenbankzwischenpeicher

Im Unterschied zu vielen anderen Anwendungen nutzt der Datenbankmanager nicht den Cache des Betriebssystems, sondern eigenen Pufferpool zum Zwischenspeichern von Daten. Da die meisten Bearbeitungsschritte für Daten in Pufferpools erfolgen, ist die Konfiguration von Pufferpools der wichtigste Einzelbereich der Optimierung.

Wenn eine Anwendung auf eine Zeile einer Tabelle zum ersten Mal zugreift, liest der Datenbankmanager die Seite, die die Zeile enthält, in den Pufferpool ein. Für weitere Anfragen sucht der Datenbankmanager zuerst im Pufferpool. Befinden sich die angeforderten Daten im Pufferpool, können sie ohne Plattenzugriff abgerufen werden, was die Anzahl der E/A-Operationen minimiert und eine höhere Geschwindigkeit ermöglicht.



# Kapitel 3

## Optimierung durch Vergleichstests

Die Durchführung von Vergleichstests (Benchmarktests) ist ein natürlicher Bestandteil des Entwicklungszyklus für Anwendungen. Nach Yevich und Lawson [YL01] die Optimierung einer Anwendung ist von Beginn an ein unerlässlicher Prozess und nur dadurch ist die Realisierung einer wirklich leistungsstarken Anwendung gewährleistet. Die Annahme, dass die Leistungsoptimierung am Ende der Anwendungsentwicklung steht, ist nach Meinung von Yevich und Lawson falsch. Die Durchführung von Vergleichstests erfordert die Zusammenarbeit von Anwendungsentwicklern und Datenbankadministratoren und sollte für eine Anwendung stattfinden, um Daten über die aktuelle Leistung zu erhalten und Ansätze zur Leistungsoptimierung zu ermitteln.

Das Ziel der Vergleichstests ist die Optimierung der Konfigurationsparameter. Sie erfordert es, SQL-Anweisungen aus der Anwendung mit variierenden Parameterwerten so lange zu wiederholen, bis die Anwendung mit der höchstmöglichen Effizienz arbeitet.

### 3.1 Vergleichstestmethoden

Bei der Durchführung von Vergleichstests ist es wichtig eine reproduzierbare zugrunde liegende Umgebung zu haben. Man sollte in der Lage sein, die Umgebung immer wieder herstellen zu können, sodass der gleiche Test unter gleichen Bedingungen durchgeführt werden kann.

Die Vergleichstests oder Messungen sollten folgende Kriterien erfüllen:

- Die Tests sind wiederholbar.
- Jede Wiederholung eines Tests beginnt mit der gleichen Systemkonfiguration.
- Die Hardware und die Software, die für die Vergleichstests verwendet werden, entsprechen der realen Umgebung.

- Es sind keine anderen Funktionen bzw. Anwendungen im System aktiv<sup>1</sup>, sofern nicht anders vorgesehen.

Für die Durchführung von Vergleichstests wird ein Szenario erstellt und die Anwendung wird mehrere Male ausgeführt, indem bei jeder Ausführung wichtige Informationen erfasst werden.

Nach [DB2-02] S.408 sollten die zu testenden SQL-Anweisungen in zwei Kategorien unterteilt werden:

- Repräsentatives SQL.– dazu zählen alle SQL-Anweisungen, die während eines typischen Einsatzes der zu testenden Anwendung ausgeführt werden und
- Extremfall-SQL.– zu dieser Kategorie gehören alle Anweisungen, die häufig ausgeführt werden, Anweisungen, die sehr große Datenmengen verarbeiten, Anweisungen, die sehr lange laufen usw.

Ein weiteres wichtiges Kriterium ist die Art des Vergleichs. Eine Möglichkeit wäre es, eine Gruppe von SQL-Anweisungen über ein Zeitintervall wiederholt auszuführen, um den Durchsatz der Anwendung zu bestimmen. Die andere Möglichkeit wäre, die erforderliche Zeit für die Ausführung Einzelner oder einer Gruppe von SQL-Anweisungen zu bestimmen. Bei dieser Diplomarbeit wird hauptsächlich die zweite Methode benutzt.

Für alle Vergleichstests wird ein effizientes Zeiterfassungssystem benötigt, um die Zeit einzelner SQL-Anweisungen oder der gesamten Anwendung zu berechnen.

Die Zeiterfassung der Ausführung von SQL-Anweisungen ist nicht der einzige wichtige Faktor bei der Leistungsanalyse. Durch sie werden nicht alle möglichen Leistungsengpässe offen gelegt. Zum Beispiel könnten Informationen über Pufferpool-ein/ausgaben darauf hinweisen, dass eine bestimmte Anweisung durch die Ein/Ausgabeaktivitäten gebremst wird. Diese Art von Daten sollte unter Umständen auch miterfasst werden.

DB2 als kommerzielle Datenbank ist mit Werkzeugen (Tools) ausgestattet, die bei der Suche nach Leistungsengpässen sehr gute Arbeit leisten. Für DB2 existieren zwei Arten von Tools: die integrierten Tools, die ohne weitere Kosten zu Verfügung stehen, und so genannte Add-Ons, die separat erworben werden müssen, aber mächtiger sind. Hier ist besonders *DB2 Performance Expert* zu erwähnen, in dem unter anderem Performance Monitoring und Pufferpool Analyse integriert sind. Da die Add-Ons kostenpflichtig sind, werden sie bei dieser Diplomarbeit nicht zum Einsatz kommen. Ausführliche Informationen über die integrierten Tools und ihre Einsatzbereiche werden in [DB2-02a] genau erklärt. Dazu gehören:

**Das Vergleichstest-Tool *db2batch*.**– Dieses Tool ermöglicht bequem die Erstellung und Auswertung eines Benchmark. Das *db2batch*-Tool setzt viele bereits erwähnte Richtlinien für die Erstellung der Vergleichstests um. Es lassen sich je nach Bedarf die Größe der

---

<sup>1</sup>Gestartete Anwendungen belegen Speicher, selbst wenn Sie auf Symbolgröße verkleinert wurden oder momentan inaktiv sind. Dadurch erhöht sich die Wahrscheinlichkeit, dass Seitenauslagerungen zu einer Verzerrung der Vergleichstestergebnisse führen, wodurch die Wiederholbarkeitsregel verletzt wird.

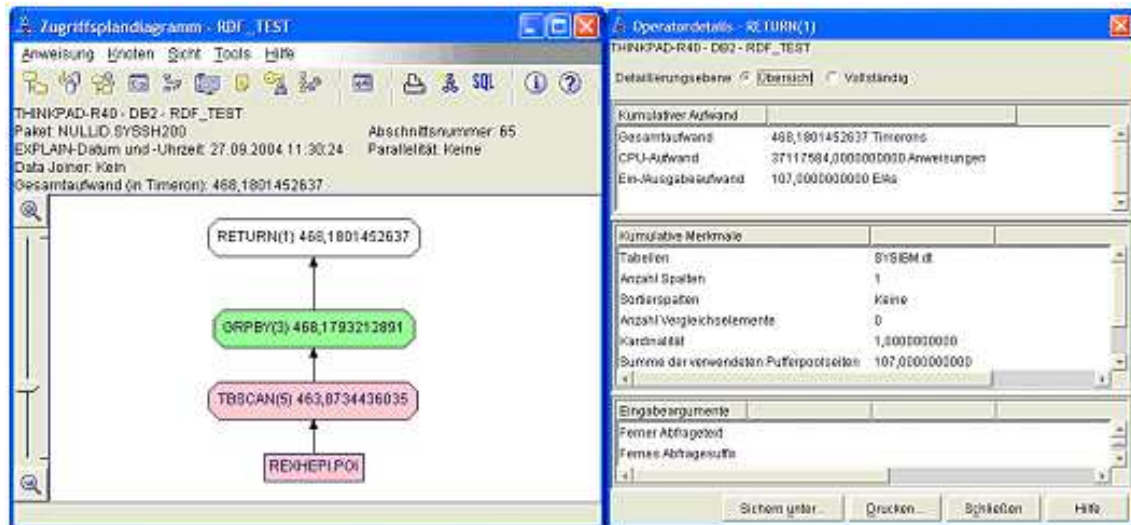


Abbildung 3.1: Screenshot von Visual Explain.

Anwortmenge und die Anzahl der Zeilen aus dieser Antwortmenge, steuern. Bei der Zeitmessung für eine Gruppe von SQL-Anweisungen berechnet *db2batch* sogar arithmetische und geometrische Mittelwerte.

**Visual Explain** ermöglicht die grafische Darstellung des Zugriffsplanes, den das DB2-Optimierungsprogramm für die Ausführung der SQL-Anweisungen ausgewählt hat. Das DB2-Optimierungsprogramm verwendet in hohem Maße die statischen Informationen über die Größe der Tabellen und Indizes der Datenbank. Außerdem werden Informationen über die Anzahl von Tupeln, die Verteilung der Daten in bestimmten Spalten von Tabellen und Indizes, die Größe des Pufferpools, die CPU-Geschwindigkeit verwendet, um den Aufwand für die Ausführung von SQL-Anweisungen abzuschätzen. Der Zugriffsplan wird dann grafisch dargestellt. Die Tabellen und Indizes und die Operationen über sie werden als Knoten dargestellt. Die Reihenfolge der durchgeführten Operationen wird durch Kanten dargestellt, sodass ein Zugriffsbaum entsteht. Dieser informiert darüber, ob z.B. die erstellten Indizes bei der Ausführung der SQL-Anweisungen berücksichtigt werden und dies zu einer Verbesserung der Leistung führt.

Neben dem Zugriffsbaum werden auch andere wichtige Informationen zur Abschätzung der Belastung zur Verfügung gestellt. Neben dem Gesamtaufwand (Total Cost), die eine Abschätzung für den Gesamtaufwand (in Timeron<sup>2</sup>) der Abarbeitung von SQL-Anweisungen darstellt, werden auch CPU-Aufwand, die Anzahl der E/A-Zugriffe und weitere Informationen angegeben. In dem mittleren Anzeigefenster werden Angaben über die Tabellen gemacht z.B. die Menge der Tabellen/Spalten, auf die zugegriffen wurde, die geschätzte

<sup>2</sup>Timeron ist die Bezeichnung für eine künstlich geschaffene Maßeinheit. Timerons werden vom DB2 Optimierer anhand interner Werte, wie z.B. gesammelte Statistikdaten, bestimmt. Die Statistikdaten ändern sich je nach Datenbanknutzung. Daher gibt es keine Garantie, dass die Timerons für eine SQL-Anweisung bei jeder Ermittlung der geschätzten Kosten in Timerons auch gleich ist.

Anzahl von Zeilen, die übergeben werden sollten (Kardinalität), die Summe der verwendeten Pufferpoolseiten usw. (Abb. 3.1).

Da es sich bei den Ergebnissen lediglich um Abschätzungen handelt, die der DB2-Optimierer auf Grund der gesammelten Katalogstatistiken vornimmt, ist es wichtig, dass nach Veränderung der Datenbankkonfigurationsparameter oder Einfügen einer großen Menge von Tupeln, die Katalogstatistiken mittels des Befehls *runstats* aktualisiert werden. Visual Explain ist ein sehr gutes und mächtiges Tool zur Analyse von SQL-Anweisungen. Der Nachteil von Visual Explain ist es, dass das Ergebnis für den Gesamtaufwand in Timerons angegeben wird und somit nicht vergleichbar ist, wenn RDF-S3 mit anderen kommerziellen oder frei erhältlichen Datenbanksystemen (MySQL, PostgreSQL usw.) getestet werden soll. Dennoch wird Visual Explain bei den Tests immer wieder verwendet, da es die Analyse von SQL-Anweisungen ohne deren „wirkliche“ Ausführung<sup>3</sup> ermöglicht.

**Momentaufnahme** (Snapshot) und **Ereignismonitor** (Event Monitor).– In DB2 es existieren zwei weitere wichtige Werkzeuge zur Analyse von Datenbankmanagern verwaltete Daten. Der Momentaufnahmemonitor zeichnet Datenbankinformationen in bestimmten Zeitintervallen auf und dient bei der Analyse von Performanceproblemen. In einem Diagramm lassen sich die gewählten Parameter im Zeitverlauf darstellen. Um Ausnahmeforderungen zu definieren, kann man Schwellenwerte (Thresholds) für Leistungsvariablen angeben. Wenn diese Werte überschritten werden, wird darauf hingewiesen. Es ist wichtig das nicht zu viele Parameter gleichzeitig und in zu kurzen Zeitintervallen abgefragt werden, denn dies könnte die Performance beeinträchtigen und somit zu einer Verzerrung der Vergleichstestergebnisse führen.

Der Ereignismonitor hingegen dient zur Aufzeichnung von Systemmonitordaten nach dem Eintreten bestimmter Ereignisse, wie zum Beispiel das Ende einer Transaktion, das Ende einer Anweisung oder die Erkennung einer gegenseitigen Sperre. Auch hier gilt: Die Erfassung sämtlicher Informationen kann die Performance der Datenbank zu stark im negativen Sinne beeinflussen.

Ausführliche Informationen zu diesen und anderen Monitorelementen sind im Handbuch: [DB2-02a] zu finden.

## Externe Tools

Mercury Interactive [MI] stellt mit LoadRunner ein Werkzeug für Belastungstests im kommerziellen Bereich zur Verfügung. Mithilfe dieses Lasttest-Tools können Tausende von Benutzern emuliert werden d.h. die virtuellen Benutzer führen auf einem System Transaktionen durch, um zum Beispiel den Datenverkehr während des Betriebes zu emulieren. Die LoadRunner Suite besteht aus verschiedenen Echtzeit-Monitoren für Applikationsserver, Datenbankserver, Webserver und Netzwerk, die präzise Performance-Messungen während

---

<sup>3</sup>Die SQL-Anweisungen werden nicht tatsächlich ausgeführt d.h. es werden keine Änderungen an Datenbanken durchgeführt.

der Lasttests durchführen. Wie bei anderen kommerziellen Tools wird LoadRunner bei dieser Diplomarbeit nicht näher behandelt.

Bei JUnitPerf [JUP] handelt es sich um eine Sammlung von JUnit-Erweiterungen [JU]. Hierbei wird Performance und Lastmessung an den bestehenden JUnit-Tests durchgeführt. Bei JUnit handelt es sich um ein Open Source Java-Testframework, das zur Entwicklung von wiederholbaren Testfällen für eine Anwendung dient. Mit JUnit ist es möglich Codestücke darauf zu testen, ob sie z.B. das erwartete Ergebnisse liefern. Durch die automatische Überprüfung des Ergebnisses kann der bis dahin erstellte Code verifiziert werden.

Durch die Erweiterung von JUnit Umgebung ermöglicht JUnitPerf die Messung und Protokollierung der abgelaufenen Zeit für die Ausführung einer definierten Aufgabe. Zudem existiert auch die Möglichkeit, einen Lasttest mit einer bestimmten Anzahl von Benutzern und Iterationen zu simulieren. Durch die Angabe von Benutzeranzahl, Dauer und Wiederholungen kann JUnitPerf das Laufzeitverhalten der Tests unter Last überprüfen. Der Einsatz von JUnit mit JUnitPerf ermöglicht bereits im frühen Stadium, die Skalierbarkeit im Kleinen zu kontrollieren.

Einem erfahrenen Java-Programmierer ist so die Möglichkeit gegeben, Lasttests zu erstellen, die sonst nur mit professionellen kommerziellen Tools, wie z.B. dem oben bereits erwähnten LoadRunner, möglich wären.

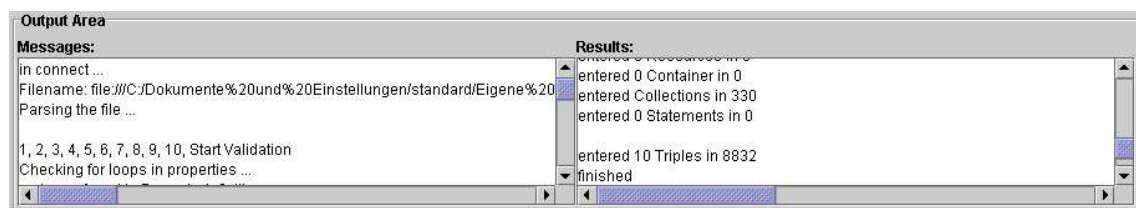


Abbildung 3.2: Die Ausgabe von RDF-S3 (Screenshot).

Schließlich bietet, RDF-S3 (Abb. 3.2) selbst die Möglichkeit der Zeiterfassung für die Ausführungsdauer von SQL-Anweisungen. Package *de.jwg.dbis.rdf.rdf3.storage* enthält die Klasse *statistic* in der die Startzeit bzw. Endzeit der Ausführung der SQL-Anweisungen mithilfe der Methoden *start\_time* bzw. *end\_time* gemessen wird, die wiederum die Messmethode *java.lang.System.currentTimeMillis* benutzen. Die Differenz der beiden Zeiten stellt die Ausführungsdauer von Anweisungen bzw. einer Gruppe von Anweisungen dar.

Ein anderer wichtiger Aspekt von RDF-S3 ist die Möglichkeit, die Ausgabe auch in Dateien zu leiten und zwar, sowohl in RDF/XML-Format als auch als reinen Text. Das ist besonders wichtig, wenn die Daten anschließend auch verarbeitet werden sollen wie z.B. in grafischer Form mit Tabellenkalkulation. Natürlich besteht keine Möglichkeit, Engpässe oder Probleme zu erkennen. Aber, da diese Funktion in die RDF-S3 Benutzeroberfläche (GUI) integriert ist, ist man unabhängig von verschiedenen Tools. Die Ergebnisse sind unabhängig vom verwendeten Datenbanksystem (DB2, MySQL usw.) jederzeit vergleichbar.

## 3.2 Ausführung von Vergleichstests – Vorgehensweise

Das Ziel dieser Arbeit ist die Performance-Optimierung der Nutzung von RDF-Daten durch RDF-S3 (RDF Source related Storage System). Eine sehr wichtige Rolle sollen dabei die Verbesserungen der Datenbankkonfigurationsparameter<sup>4</sup> und die Festlegung der Vorgehensweise spielen.

### Standarddatenbank

Bei der Erstellung einer neuen Datenbank (in unserem Fall ist die RDF\_S3 Datenbank) wird der Benutzer von DB2 mit einem Assistenten unterstützt. Dieser schlägt in Abhängigkeit von der vorhandenen Hardware eine Reihe von Standardeinstellungen vor, die ohne großen Aufwand erledigt werden können.

Es wird ein Tabellenbereich für die Benutzer-, Systemkatalog- und temporären Systemtabellen angelegt. Dieser wird standardmäßig als SMS (System Managed Space) deklariert, da in diesem Fall der Bereich automatisch vergrößert wird. Die *extentsize* und *prefetchsize* lassen sich bei der Deklaration des Tabellenbereichs als SMS nicht ändern und haben die Standardwerte 16 4-KB Seiten<sup>5</sup> und 16 4-KB Seiten. Der Pufferpool (*buff\_page*-Parameter) wird mit der Standardgröße von 250 Seiten initialisiert. Außerdem ist die Anzahl asynchroner Seitenlöschfunktionen (*num\_iocleaners*) bzw. E/A-Servers (*num\_ioservers*) auf 1 bzw. 3 gesetzt. Die Größe des gemeinsamen Speichers der Datenbank (*database\_memory*) ist dem System überlassen d.h. der Parameter ist auf *automatic* gesetzt und wird vom System auf 6560 Seiten (26240 KB = 25,625 MB) gesetzt.

Die so erstellte Datenbank (RDF\_S3) ist die so genannte Standarddatenbank. Bevor es zu Messungen kommt, wird ein BACKUP gemacht. Die Datenbank kann jederzeit mit RESTORE in dem Anfangszustand versetzt werden.

Zuerst wird ein Konfigurationsparameter ausgewählt und dann wird ein Test mit verschiedenen Parameterwerten ausgeführt, bis die maximale Leistungssteigerung erzielt wird. Anschließend wird die Anwendung mit demselben Parameterwert zehnmal<sup>6</sup> (damit die Auswirkungen der Änderung von Parameterwerten besser einzuschätzen sind) ausgeführt, um einen Durchschnittswert für die benötigte Zeit zu ermitteln. Der erste Durchlauf, der so genannte Aufwärmdurchlauf (Warm-up Run) wird nicht berücksichtigt, wie in [DB2-02] empfohlen. Da der Aufwärmdurchlauf einige Startaktivitäten wie z.B. die Initialisierung

---

<sup>4</sup>Siehe Anhang A

<sup>5</sup>Eine Seite ist 4 KB groß und wird, wenn nicht anders angegeben, als Standardwert verstanden.

<sup>6</sup>Aus Zeit Gründen wurde die Anzahl der Iterationen von ursprünglich zwanzig auf zehn zurückgenommen, da die Ergebnisse verschiedener Iterationen minimal abwichen. Für die großen RDF-Dateien wurde die Zahl der Wiederholungen nochmal auf drei reduziert.

des Pufferpools, durchführt, dauert er etwas länger als die anderen so genannten Normaldurchläufe (Normal Run).

### 3.3 Die Testumgebung

Damit die Tests nachvollziehbar sind, wird in diesem Abschnitt die zugrunde liegende Hard- und Software der Testumgebung vorgestellt. Es ist nicht auszuschließen, dass nach erneuter Durchführung der Testreihen andere Testergebnisse gewonnen werden können. Es ist aber nicht davon auszugehen, dass sich die neuen Testergebnisse im Wesentlichen von den hier gewonnenen unterscheiden.

Für die Tests dient ein einziger Rechner sowohl als Server als auch als Client d.h. bei dieser Arbeit war es nicht möglich, die Tests über das Netzwerk durchzuführen, um die reale Umgebung besser zu simulieren. Dennoch sind die gewonnenen Ergebnisse durchaus als repräsentative Ergebnisse zu sehen.

Bei dem Datenbankmanagementsystem handelt es sich um die Version 8.1 der Personal Edition von IBMs UDB DB2. Die Personal Edition (PE) hat alle Eigenschaften von DB2 Workgroup Server Edition – mit einer einzigen Ausnahme: Die fernen Clients können nicht mit der Datenbank, die auf PE läuft, Verbindungen herstellen. Dennoch werden alle Anwendungen, die für DB2 PE entwickelt wurden, auch auf alle anderen DB2 Editionen laufen. Auf dem Testrechner ist MS Windows XP Professional (SP1) installiert.

#### Die Hardware

Der Server (Client), auf dem die Datenbankinstanz läuft, ist ein IBM ThinkPad R40. Die CPU des Rechners ist ein Intel Pentium M<sup>®</sup> Prozessor mit 1.4 GHz Taktfrequenz. Die 512 MB RAM Arbeitsspeicher sind vielleicht knapp bemessen, aber für unsere Zwecke sollten sie ausreichen. Dafür werden einige unnötige Prozesse (Programme) deaktiviert, sodass zusätzlich ca. 45 MB RAM „gewonnen“ werden können. Die eingebaute Festplatte ist zwar mit 4000 U/Min nicht sehr schnell – da wäre eine SCSI-Platte von Vorteil gewesen – aber mit 36 GB groß genug für unsere Tests.

In der Abschlussphase stand uns zusätzlich noch ein zweiter Rechner zu Verfügung. Die 512 MB RAM Arbeitsspeicher waren für die meisten Tests ausreichend, aber die CPU (Intel Pentium III 1.0 GHz) war für unsere Zwecke doch ein wenig langsam. Die Ausführungsdauer von SQL-Anweisungen stieg dementsprechend. Dennoch ergab sich mit dem zweiten Rechner die Möglichkeit, viele verschiedene Konfigurationen wie z.B. Verteilung der Tabellenbereiche auf zwei Festplatten zu testen. Die erste Festplatte ist 36 GB groß und hat eine Drehgeschwindigkeit von 4000 U/Min. Die zweite Festplatte hingegen ist 72 GB groß und hat eine deutlich höhere Drehgeschwindigkeit – 7200 U/Min. Dennoch blieb der erste Rechner das Hauptwerkzeug für unsere Tests, da sonst viele Tests auf dem zweiten Rechner hätten wiederholt werden müssen, was aus Zeitgründen nicht möglich war.



# Kapitel 4

## Optimierung der Konfigurationsparameter

In diesem Kapitel werden zunächst die Testdaten und die beiden Testmethoden vorgestellt. Im Anschluss daran wird die Entwicklung von RDF-S3 kurz angesprochen. Danach werden die Konfigurationsparameter des Datenbanksystems, die zur Steigerung der Performance von RDF-S3 eingesetzt werden können, analysiert und getestet. Am Ende dieses Kapitels folgt der Vergleich der Standardkonfiguration mit der optimierten Konfiguration.

In den Abschnitten 4.1-4.11 basiert das Speichern der RDF-Daten auf der Erzeugung des Speicher-Modells mithilfe von VRP. Im Abschnitt 4.12 hingegen wird die *Stream Based Parsing*-Methode benutzt. Dabei werden die RDF-Tripel direkt gespeichert (Siehe Kap. 2.2 auf S.15).

### 4.1 Die Testdaten und Methoden

Um sinnvolle aussagekräftige Testergebnisse zu bekommen, war es wichtig die Eingabe d.h. die RDF-Daten so zu wählen, dass RDF-S3 als Anwendung voll ausgelastet wird. Wie in Kapitel 2.2 erwähnt, muss die Datenbank initialisiert werden, bevor mit den Speichern von RDF-Daten begonnen werden kann. Danach können die RDF-Daten mittels RDF-S3 gespeichert werden. Die RDF-Datei wird zuerst geparkt und anhand der gewonnenen Informationen wird das Schema erzeugt (Dabei werden Tabellen erzeugt, die dann die eigentlichen Daten aufnehmen). Um realistische Testwerte zu bekommen, wurden als Eingabe verschiedene RDF-Dateien in variierenden Größen gewählt: RDF-Datei *culture-data.rdf* (4 KB), *chefmoz.guides-mod.rdf*<sup>1</sup> (242 KB) und *PerfTest.rdf* (1462 KB). Ein anderer wichtiger Aspekt war es, der Anwendung RDF-S3 als Eingabe Schema-Daten (RDF-Dateien, die nur das Erzeugen von Schema zufolge haben) zu geben. Dabei werden nur die Benutzertabellen erzeugt<sup>2</sup>, die dann Daten aus der RDF-Datei *culture-data.rdf* aufnehmen. Für diese Tests

---

<sup>1</sup>Diese Datei ist die modifizierte RDF-Datei *chefmoz.guides.rdf*.

<sup>2</sup>Dabei werden dennoch einige Datensätze gespeichert. In der Tabelle sources wird ein Eintrag von allen benutzten Dateien gespeichert.

wurde die RDF-Datei *culture.rdf* (3 KB) ausgewählt. Je nach Datei war z.B. auch die Anzahl der RDF-Tripel, die gespeichert wurden, sehr verschieden:

- *culture-data.rdf* (4 KB) mit 51 RDF-Tripel
- *chefmoz.guides-mod.rdf* (242 KB) mit 4011 RDF-Tripel
- *PerfTest.rdf* (1462 KB) mit 37500 RDF-Tripel und
- *culture.rdf* (3 KB) mit 40 RDF-Tripel.

Für die Schema-Datei *culture.rdf* ist hier insbesondere die Anzahl von *classes* – 12 und *properties* – 13 zu erwähnen, da sie die Erzeugung von zusätzlichen Tabellen zur Folge hat – 21 neue Tabellen wurden somit angelegt.

### 4.1.1 Testmethoden

Sowohl für RDF-Daten als auch für RDF-Schema-Daten war es wichtig, das Verhalten von RDF-S3 über längere Zeit zu beobachten. Die einzelnen Tests werden auf leere Datenbank, die jeweils durch RESTORE in einen vordefinierten Zustand versetzt wird, ausgeführt (Siehe Kap. 3.2). Da die einzelnen Tests mit unterschiedlichen Größen auch unterschiedlich lang dauerten, wurde die Anzahl der Dateien bzw. Wiederholungen unterschiedlich gewählt:

1. 1500 für die kleinen RDF-Dateien (4 KB),
2. 100 für die mittelgroßen RDF-Dateien (242 KB) und
3. 10 für die großen RDF-Dateien (1462 KB).

Dabei wurden zwei Testreihen durchgeführt. Die erste Testreihe bestand aus einzelnen Dateien, während bei der zweiten Testreihe die Performance Test Methode von RDF-S3 (Menüeintrag Database → Performance Test) benutzt wurde. Da aber die einzelnen Tests sehr viel Zeit in Anspruch nahmen und z.B. ein einzelner Test für die RDF-Dateien mittlerer Größe ca. 3 Stunden dauerte, wurde die Anzahl der RDF-Dateien von 1500 für die kleinen Dateien bzw. 100 für die mittelgroßen Dateien auf 500 bzw. 50 reduziert. Die Tests für die großen RDF-Dateien wurden nur für die wichtigsten Konfigurationsparameter durchgeführt, da ein einzelner Test mit 10 einzelnen RDF-Dateien über 8 Stunden dauerte. Für die Standarddatenbank und die optimierte Datenbank wurden die Tests mit der höchsten Anzahl der Wiederholungen ausgeführt (1500, 100 und 10).

Der Unterschied zwischen beiden Methoden besteht darin, dass bei der Performance Test Methode die RDF-Datei nur einmal zu Beginn des Tests vom VRP geparkt wird und die gewonnenen Informationen dann für folgende Iterationen zur Verfügung stehen, während bei der anderen Methode die RDF-Datei jedes Mal geparkt wird. Diese Methode der Bearbeitung jeder einzelnen Datei entspricht eher der Realität, aber um die Ausführungsdauer einzelner Tests nicht zu lang werden zu lassen und den entsprechenden

zeitlichen Aufwand bewältigen zu können, wurde die Performance Test Methode bevorzugt. Die Tests zeigten, dass das Verhalten von RDF-S3 mit der Performance Test Methode geringfügig anders war. Der Anstieg der Kurve bei der Performance Test Methode ist flacher als bei der anderen Methode (Abb. 4.1). Der Unterschied beider Methoden ist darauf zurückzuführen, dass bei der Methode mit einzelnen Dateien, im Vergleich zu der Performance Test Methode mehr Seitenfehler (*page faults*<sup>3</sup>) beim Parsen auftreten, da die RDF-Dateien jedes Mal neu geparkt werden müssen. Die Tests wurden sowohl für die Standarddatenbank (SMS) als auch für die Datenbank mit DMS-Tabellenbereichen durchgeführt. Die blaue Linie zeigt den Verlauf der Zeiten für die Standarddatenbank mit der Methode von einzelnen Dateien. Die grüne Linie stellt den Verlauf des Tests mit der Performance Test Methode dar.

Die Beschriftung der y-Achse zeigt die Zeit in Millisekunden, die für das Speichern aller

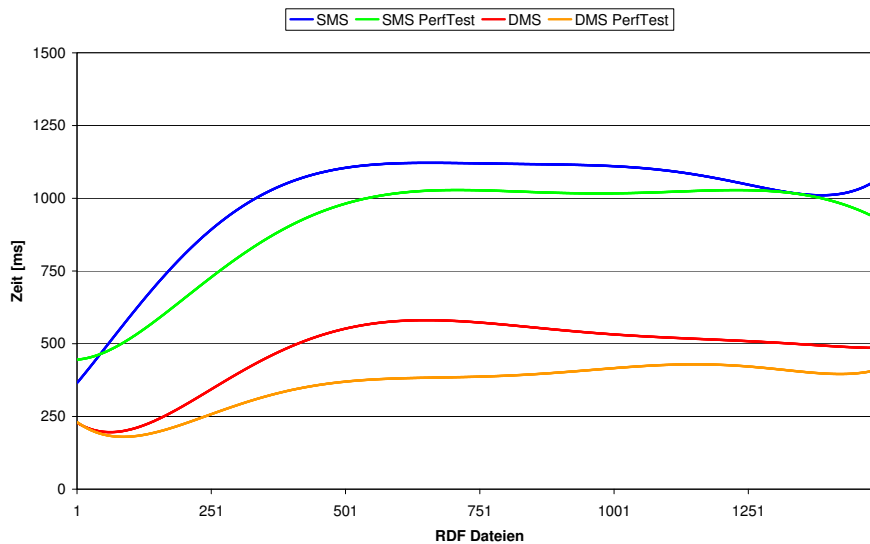


Abbildung 4.1: Vergleich der beiden Testmethoden.

RDF-Tripel einer RDF-Datei nötig war. Die x-Achse zeigt die Anzahl der RDF-Dateien, die für einen einzelnen Test benutzt wurden. Um die Unterschiede zwischen den verschiedenen Versionen für den Betrachter einfacher sichtbar zu machen, wurde der Verlauf der Zeiten für die RDF-Dateien kleiner Größe als Trendlinie<sup>4</sup> dargestellt. Die folgende Abb. 4.2 verdeutlicht dieses Konzept.

In der Farbe Blau sind die eigentlichen Zeiten für einzelne RDF-Dateien dargestellt. Die auftretenden großen Unterschiede zwischen den einzelnen Zeiten sind darauf zurückzuführen, dass zum einen die Tabellenbereiche als SMS deklariert wurden und somit das

<sup>3</sup>Ein Seitenfehler tritt auf, wenn die angeforderte Seite im Hauptspeicher nicht vorhanden ist.

<sup>4</sup>Um die Diagrammkurve glatter erscheinen zu lassen, wurde die im Tabellenkalkulationsprogramm integrierte Trendlinienfunktion benutzt.

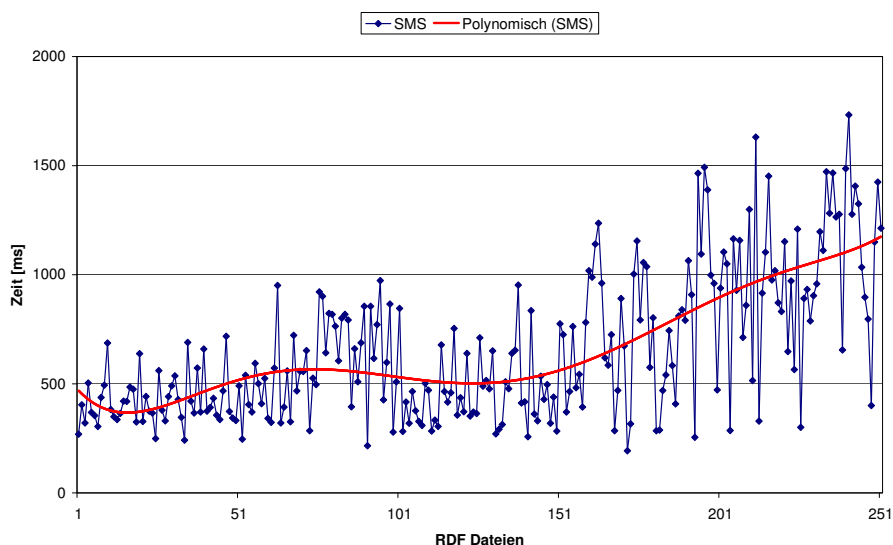


Abbildung 4.2: Test mit RDF-Dateien kleiner Größe und SMS als Tabellenbereich.

Betriebssystem für das Speichern der Daten verantwortlich war, was je nach Aktivität anderer Prozesse zu einer Verschiebung der Zeiten nach Oben oder nach Unten zur Folge hatte. Zum Zweiten waren die Zeiten der Ausführungsdauer zu kurz (weniger als 1 Sekunde) und somit hatte jede kleine Aktivität anderer interner Betriebssystemprozesse große Auswirkungen. In den nächsten Abschnitten werden wir sehen, dass zum einen die Schwankungen bei den Tests mit RDF-Dateien mittlerer Größe und großen RDF-Dateien kaum zu beobachten sind und zum anderen bei den DMS-Tabellenbereichen entscheidend kleiner sind, da für das Speichern der Daten DB2 verantwortlich ist und somit Betriebssystemmechanismen umgegangen werden.

Um den Performancegewinn zwischen zwei Testläufen zu berechnen, wird zunächst für jede einzelne Datei der Zeitunterschied beider Testläufe gemessen. Dieser Wert wird dann in Prozent umgerechnet, um anschließend einen Mittelwert von allen Dateien eines Testlaufes zu bilden.

Zunächst wurde eine Testreihe durchgeführt, die den Unterschied der Performancesteigerung verdeutlicht, wenn Veränderungen an der verwendeten Hardware vorgenommen werden (Abb. 4.3). Der erste Test wurde auf einer Festplatte mit 4000 U/Min (blaue Linie) durchgeführt, wobei alle Tabellenbereiche als SMS deklariert wurden. Der zweite Test wurde auf einer anderen schnelleren Festplatte – 7200 U/Min – wiederholt. Die Abb. 4.3 zeigt den Test mit RDF-Dateien mittlerer Größe und zeigt, dass sich durch Hardwareaufrüstung z.B. der Festplatte, enorme Performancesteigerungen erzielen lassen. Die Zeiten sind von ca. 42 Sekunden am Anfang auf knapp über 30 Sekunden gesunken, um am Ende sogar von ca. 70 Sekunden auf knapp unter 40 Sekunden zu fallen, was eine Performancesteigerung von ca. 38 % im Durchschnitt bedeutet. Die Hardwareaufrüstung ist immer empfehlens-

wert, wenn die anderen Maßnahmen nicht das erwünschte Ergebnis liefern.

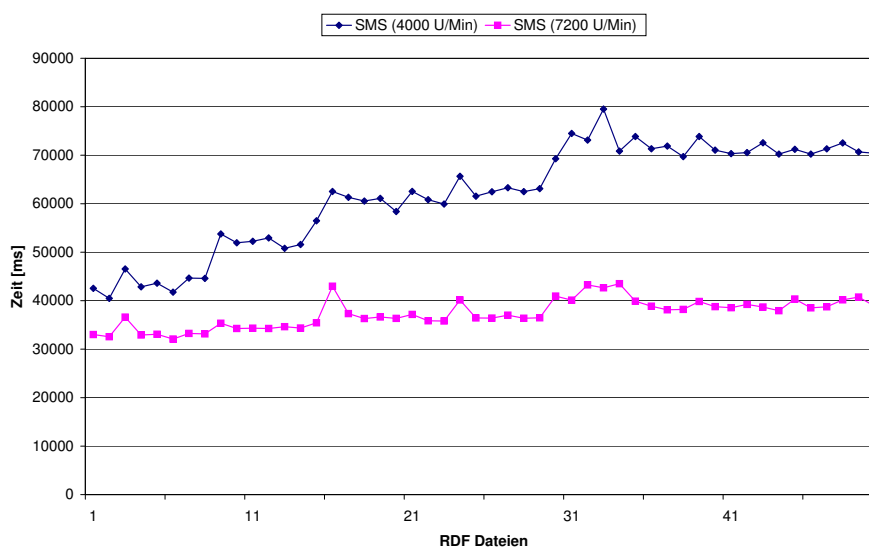


Abbildung 4.3: Auswirkungen einer Hardwareaufrüstung (Festplatte) auf die Laufzeiten.

## 4.2 RDF-S3 Entwicklung

Zu Beginn dieser Arbeit lag RDF-S3 in der Version 1.6 (veröffentlicht am 05. Oktober 2004) vor. Da die Entwickler sehr bemüht waren, neue sinnvolle Features einzubauen, hat sich auch die zugrunde liegende RDF-S3 während dieser Arbeit einige Male geändert. Zuerst wurden in einer Pre-Release Version 1.7 (17. Dezember 2004) einige Bugs von Version 1.6 beseitigt. In der Version 1.7 (07. Januar 2005) wurde zusätzlich noch die Optimierung der Cachegröße für die *Namespaces* und *Recources* Variablen<sup>5</sup> durchgeführt, was auf die Performance von RDF-S3 Auswirkung zeigte. Durch Tests erwiesen sich die Werte 100 für die *Namespaces* und 10000 für *Recources* als angemessene Werte. In der Version 1.7 ist es zusätzlich möglich, diese Werte auch interaktiv über GUI (Menüeintrag Database → Cache Sizes) eigenen Bedürfnissen anzupassen. Außerdem enthält die Version 1.7 von RDF-S3 die Möglichkeit, die Anzahl der Wiederholungen interaktiv durch den Benutzer (Menüeintrag Database → Performance Test) zu ändern, was sich für unsere Zwecke als sehr hilfreich erwies.

Durch die Weiterentwicklung von RDF-S3 während dieser Arbeit, ergab sich die Möglichkeit, die verschiedenen Versionen von RDF-S3 vergleichend zu testen, was nicht die Hauptaufgabe dieser Diplomarbeit ist. Dennoch wurden am Anfang Tests mit der Standarddatenbank für die drei Versionen (1.6, 1.7Pre und 1.7) durchgeführt. Die Tests bestätigten

<sup>5</sup>*Namespaces* und *Recources* Variablen definieren die Größe des Cache zum Zwischenspeichern von Namespaces und Resources.

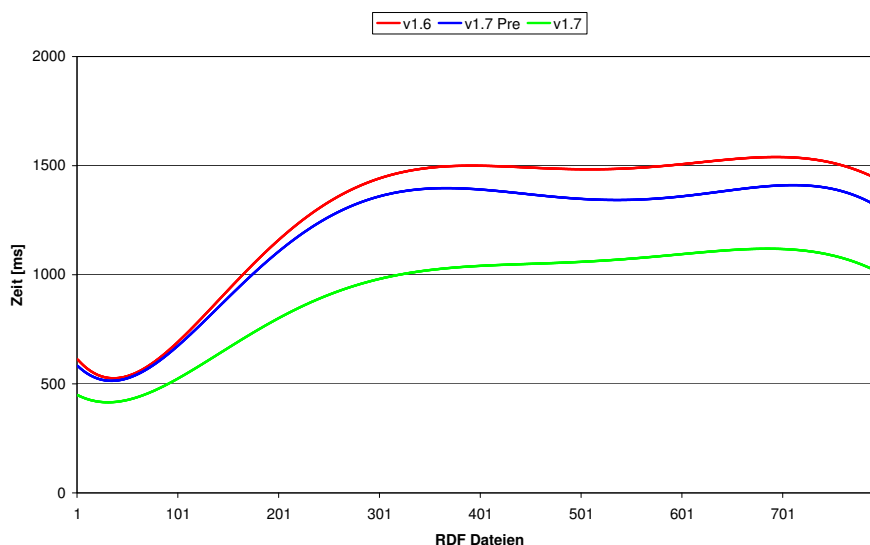


Abbildung 4.4: Vergleich zwischen verschiedenen Versionen von RDF-S3.

die erhoffte Performancesteigerung von RDF-S3 von Version zu Version (Abb. 4.4). Aus Zeitgründen wurde dieser Vergleich dann nicht weiter verfolgt. Dennoch ist es offensichtlich, wie die Abb. 4.4 zeigt, dass die Version 1.7 (hier in Grün dargestellt) effizienter als die zwei anderen Versionen (1.6 in Rot und 1.7Pre in Blau) arbeitet – im Durchschnitt ca. 27 % besser als die Version 1.6.

Bevor mit der Durchführung von Tests angefangen wird, soll die Änderung des vor-konfigurierten Wertes von *Recources Cache* Variable (vordefinierter Wert ist 10000) näher betrachtet werden. Aus den Tests mit großen Dateien fiel auf, dass für diese Tests der Wert der *Recources Cache* Variable nicht ausreichend groß war. In der RDF-S3-Ausgabe wird darauf hingewiesen, dass es zu Überläufen des Cache gekommen ist<sup>6</sup>. Die Neuanpassung des Wertes für die *Recources Cache*-Variable von 10000 auf 40000 zeigte eine Performancesteigerung von ca. 35 % im Durchschnitt (Abb. 4.5 auf S.45 zeigt die Ergebnisse mit SMS-Tabellenbereiche). Bei den Tests mit DMS-Tabellenbereichen fiel die Performancesteigerung geringer als bei SMS aus, und zwar ca. 33 % im Durchschnitt.

Da es bei den Tests mit kleinen und mittelgroßen Dateien keine Überläufe des Cache gab, war es nicht notwendig, eine Anpassung vorzunehmen. Ein Test mit mittelgroßen Dateien und einem Wert von 40000 für die *Recources Cache*-Variable zeigte ebenfalls eine Performancesteigerung. Daher ist es immer empfehlenswert, diese Werte im normalen alltäglichen Einsatz von RDF-S3 durch Tests eigenen Bedürfnissen anzupassen.

<sup>6</sup>Die Meldung *resource cache cleared* weist darauf hin.

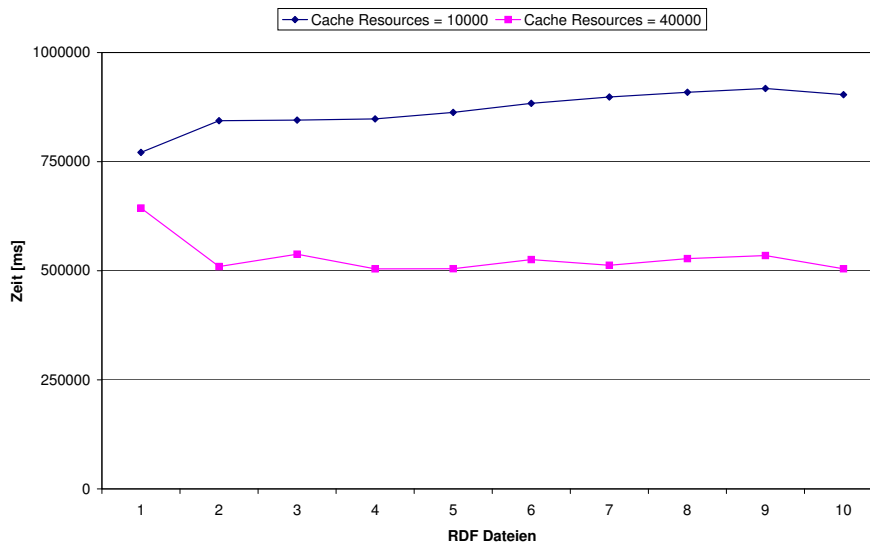


Abbildung 4.5: Die Anpassung des Wertes der *Cache-Resources* Variable.

### 4.3 SMS vs. DMS

Wie bereits in früheren Abschnitten erläutert, handelt es sich hier um Alternativen zur Speicherung von Daten auf der Festplatte. Die Daten werden in Tabellenbereiche<sup>7</sup> gespeichert. In Abschnitt 3.2 wurde die Erstellung der Standarddatenbank, die standardmäßig vom Datenbankmanager als SMS deklariert wird, erläutert, d.h. die Speicherung von Daten wird dem Betriebssystem überlassen. Für jede Datenbank werden drei Tabellenbereiche erzeugt:

- USERSPACE1.- für Benutzertabellen
- SYSCATSPACE.- für die Systemkatalogtabellen<sup>8</sup> und
- TMPSPACE1.- für die temporären Systemtabellen<sup>9</sup>

Für die Standarddatenbank werden alle drei Tabellenbereiche als SMS deklariert. Ohne die Verteilung der Daten in Tabellenbereiche und deren genaue Konfiguration zu verändern, wurde die Datenbankkonfiguration so variiert, dass statt der SMS-Tabellenbereiche DMS-Tabellenbereiche verwendet wurden. Dabei wurde in einer Testreihe der Versuch gemacht nur den Tabellenbereich für Benutzertabellen (USERSPACE1) als DMS zu deklarieren und dann in einer anderen Testreihe auch die Systemkatalogtabellen (SYSCATSPACE) als

<sup>7</sup>Siehe Abschnitt 2.3.2.

<sup>8</sup>Die Systemkatalogtabellen enthalten Informationen wie z.B. Beschreibungen von Indizes, Tabellen usw.

<sup>9</sup>Der temporäre Speicherbereich wird u.a. für Sortieren und Reorganisieren von Tabellen, Erstellen von Indizes und Verknüpfung von Tabellen verwendet.

DMS zu deklarieren. Da die temporären Systemtabellen variablen Speicherbedarf haben können, werden sie als SMS deklariert. Damit ist das Betriebssystem für die Verwaltung des Speichers verantwortlich und kann nach Bedarf den Speicher dynamisch vergrößern. Die Deklaration der temporären Systemtabellen als DMS-Tabellenbereich würde zu viel Aufwand mit sich bringen, da beim DMS-Tabellenbereich schon bei der Erstellung die endgültige Größe festgelegt werden muss. Eine andere Möglichkeit wäre es, den Tabellenbereich ausreichend groß zu dimensionieren, wobei u.U. viel Speicherplatz verschenkt werden könnte. In Zeiten der immer billiger werdenden Speichermedien ist dieser Nachteil vielleicht nicht der entscheidende Faktor, aber das Verschwenken von Speicherplatz widerspricht der Regel der optimalen Ressourcennutzung.

Bei der Auswertung der Testergebnisse zeigte sich, dass die DMS-Tabellenbereiche in allen Lagen den SMS-Tabellenbereichen vorzuziehen sind. Bei den Tests mit kleinen RDF-Dateien und Dateien mittlerer Größe betrug der Performancegewinn ca. 50 % bzw. ca. 55 % (Abb. 4.6 und Abb. 4.7), aber bei den Tests mit großen Dateien, war der Performancegewinn lediglich ca. 3 % im Durchschnitt (Abb. 4.8). Das ist darauf zurückzuführen, dass bei den Tests mit den großen Dateien andere Faktoren wie Hauptspeicherbegrenzung die Performancesteigerung bremsen.

Außerdem zeigten die Tests, dass die Deklaration von Benutzertabellen und Systemkatalogtabellen als DMS bessere Ergebnisse liefert als wenn nur die Benutzertabellen als DMS deklariert werden.

Man beachte: wie in [iX 3/05] gezeigt, können SMS-Tabellenbereiche bessere Ergebnisse als DMS-Tabellenbereiche liefern. Dabei ging es um die Sicherung bzw. Wiederherstellung der Datenbank (BACKUP und RESTORE) mit einem RAID-System. Weitere Informationen zu diesem Thema sind in [iX 3/05] zu finden.

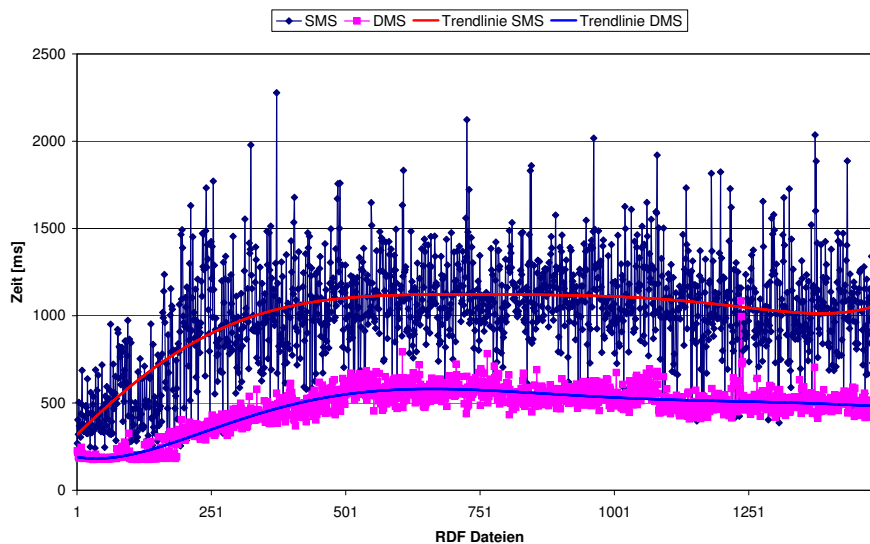


Abbildung 4.6: Der Vergleich von SMS mit DMS – kleine Dateien.

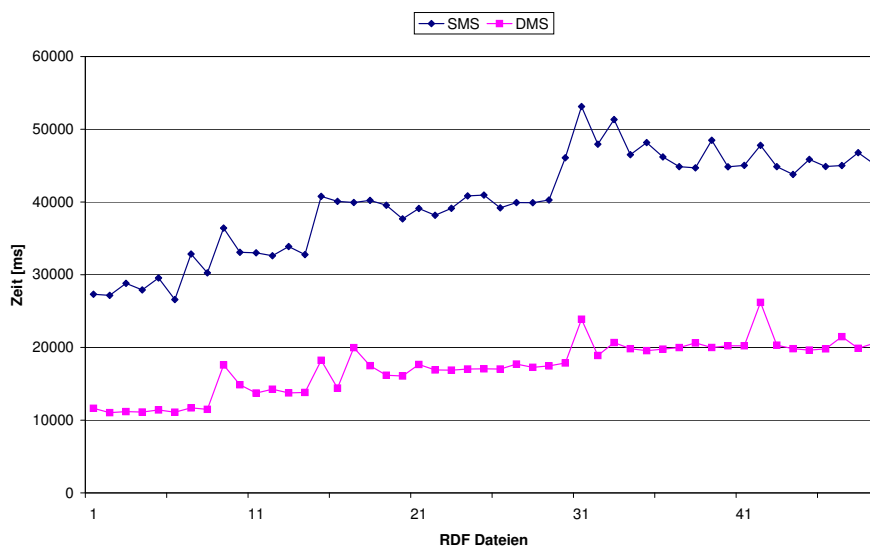


Abbildung 4.7: Der Vergleich von SMS mit DMS – mittelgroße Dateien.

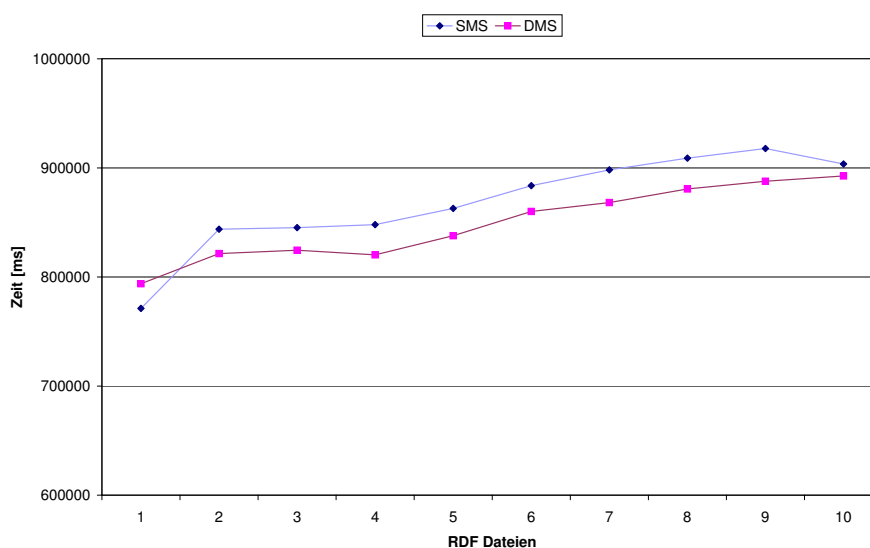


Abbildung 4.8: Der Vergleich von SMS mit DMS – große Dateien.

## 4.4 Die Tabellenbereiche

Die Anzahl und Konfiguration der Tabellenbereiche beeinflussen die Performance, da der Tabellenbereich eine sehr stark belastete logische Komponente der Datenbank ist. Beim Entwurf der Tabellenbereichstruktur ist aber immer die einfache Wartung im Auge zu behalten. Die Festlegung der Größe eines Tabellenbereichs ist bei den hier durchgeführten Tests wegen der eher geringen Menge an Daten nicht von allzu großer Bedeutung. Den-

noch sollte man bei der Erstellung der Datenbanken die Tabellenbereiche hinreichend groß dimensionieren.

#### 4.4.1 Die Konfiguration der Tabellenbereiche

In diesem Abschnitt werden die beiden wichtigsten Konfigurationsparameter der Tabellenbereiche, die *extentsize* und *prefetchsize*, optimiert.

Der *Extentsize*-Parameter bezeichnet die Anzahl der Datenseiten, die in einen Behälter des Tabellenbereichs geschrieben werden, bevor zum nächsten Behälter gewechselt wird. Bei DMS-Tabellenbereichen können Werte zwischen 2 und 256 angegeben werden, allerdings nur bei der Erstellung des Tabellenbereichs. Eine spätere Änderung des Wertes ist nicht möglich.

Der *Prefetchsize*-Parameter bezeichnet die Anzahl der Datenseiten, die der Datenbankmanager bei einem Vorablesezugriff gleichzeitig liest. Der Wertebereich dieses Parameters liegt zwischen 4 und 32767. Er kann im Gegensatz zu *extentsize* auch online d.h. im laufenden Betrieb geändert werden. Es ist z.B. angebracht, den Wert von *prefetchsize* zu verringern, wenn der Pufferpool von Seitenlöschfunktionen immer wieder geleert werden muss, weil es zu wenige Treffer im Pufferpool gibt. D.h. es werden zu viele Daten von den Vorablesefunktionen in den Pufferpool transferiert, aber nur wenige davon werden gebraucht.

Die beiden Werte sollten nicht isoliert voneinander optimiert werden, da der Wert von *prefetchsize* als ein Vielfaches von *extentsize* gewählt werden sollte [DB2-02, S.311], um mithilfe von Vorablesefunktionen mehrere durch *extentsize* definierte Bereiche parallel lesen zu können.

In einer ersten Testreihe wurde der Versuch gemacht, nur die *prefetchsize* zu ändern, wobei zunächst der Standardwert von 16 Seiten getestet wurde. Anschließend folgten Tests mit dem Wert von 64 Seiten. Der Wert von *extentsize* blieb zuerst auf dem Standardwert von 32 Seiten. Bei allen Tests mit verschiedenen Werten von *prefetchsize* konnten keine messbare Performanceverbesserungen festgestellt werden. Dies könnte daran liegen, dass das Erhöhen des *prefetchsize* Wertes alleine nicht ausreicht, um die Vorteile des Auslesens von großen Datenmengen aus einer Tabelle zum Vorschein kommen zu lassen.

Wie bereits erwähnt, sollte die Optimierung beider Parameter nicht isoliert voneinander betrieben werden. Daher folgte eine zweite Testreihe mit dem Wert *extentsize*=64 und *prefetchsize*=64. Bei beiden Testreihen zeigte sich eine kleine Steigerung der Performance.

Bei den Tests mit kleinen Dateien war die Performancesteigerung am deutlichsten. Dies ist darauf zurückzuführen, dass die einzelnen Zeiten unter eine Sekunde liegen und sich somit eine kleine Verbesserung stärker auf die Gesamtzeiten auswirkt. Im Durchschnitt war der Performancegewinn ca. 5 %. Bei den Tests mit mittelgroßen Dateien fiel die Performancesteigerung kleiner aus, im Durchschnitt ca. 3 %.

Die Abb. 4.9. zeigt die Verbesserung der Performance bei den Tests mit großen Dateien.

Das Erhöhen des *prefetchsize* Wertes (lila Linie) allein brachte keine nennenswerte Verbesserung. Die zusätzliche Abstimmung der *extentsize* Variable (gelbe Linie) dagegen brachte eine kleine Verbesserung, die jedoch im Durchschnitt nur ein knappes Prozent betrug. Je länger der Test dauerte, desto größer wurde auch der Performancegewinn (max. 3 %). Dies ist darauf zurückzuführen, dass die Datenbank am Anfang nicht so viele Datensätze enthält und somit die Änderungen an *extentsize* und *prefetchsize* keine Performancesteigerung erzielen können. Eine längere Beobachtung dieses Verhaltens war im zeitlichen Rahmen dieser Diplomarbeit nicht möglich. Das Ausbleiben der erhofften höheren Performancesteigerung ist wohl darauf zurückzuführen, dass die verschiedenen Tabellen sämtlich demselben Behälter (USERSPACE1 Tabellenbereich) zugeordnet waren und daher die unterschiedlichen Anforderungen der Tabellen nicht erfüllt werden konnten. Diese Frage soll im folgenden Abschnitt näher betrachtet werden, in dem die Zuordnung einzelner Tabellen zu den Tabellenbereichen untersucht wird.

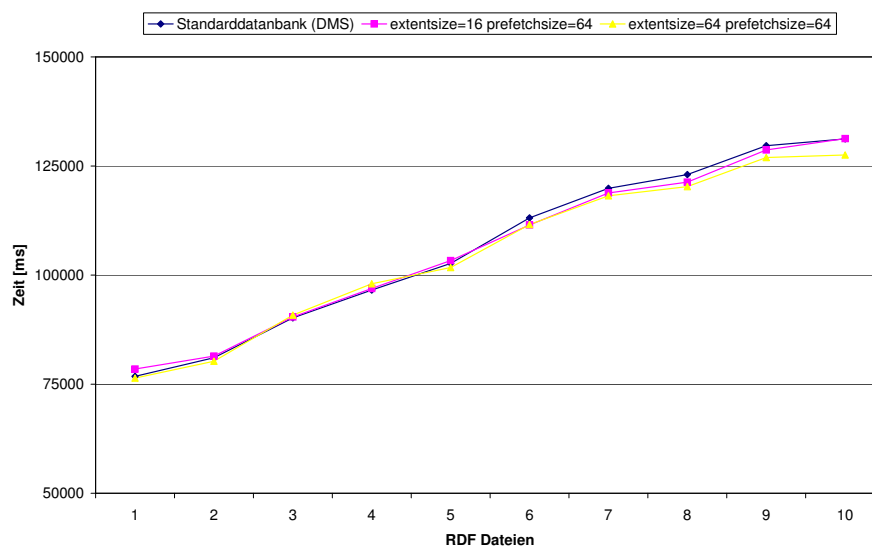


Abbildung 4.9: Konfigurationsparameter *prefetchsize* und *extentsize*.

## 4.5 Datenverteilung auf mehrere Tabellenbereiche

Die Benutzertabellen können verschiedenen Tabellenbereichen zugeordnet werden. Die einfachste Variante ist, wenn bei der Erstellung der Datenbank der Standardtabellenbereich USERSPACE1<sup>10</sup> als SMS-Tabellenbereich benutzt wird, was für unsere Zwecke nicht zufriedenstellend ist. Da sich bei den SMS-Tabellenbereichen, selbst bei der Erstellung, die *extentsize*-Größe nicht ändern lässt, fällt die Wahl zwangsläufig auf DMS-Tabellenbereiche. Eine andere Lösung wäre es, für jede Tabelle einen eigenen Tabellenbereich zu erzeugen.

<sup>10</sup>USERSPACE1 ist der Standardtabellenbereich für die Benutzertabellen.

Dies ist nicht zu empfehlen, denn die Anzahl der Tabellen nach der Initialisierung der Datenbank wäre mit 28 Benutzertabellen recht hoch. Zusätzlich werden je nach Bedarf von RDF-S3 automatisch andere Benutzertabellen erzeugt, womit die Anzahl der Benutzertabellen weiter steigt. Wenn die Datenbank dennoch eine optimierte Performance nachweisen soll, muss auch die Zuordnung der Tabellen zu den Tabellenbereichen optimiert werden.

Ein geeigneter Tabellenbereich ist dann optimal, wenn er nicht zu groß ist und auf der Festplatte kein Platz verschenkt wird. Er muss aber auch die Anforderungen der enthaltenen Tabellen erfüllen. Bei häufig veränderten Daten empfiehlt sich, für *extentsize* einen höheren Wert als der von DB2 empfohlenen Wert von 16 Seiten zu wählen. Dies würde häufiges Wechseln von Behältern beim Schreiben vermeiden.

Für die Tests war es wichtig, die Benutzertabellen so zu verteilen, dass nicht zu viele Tabellenbereiche entstehen, denn die Tabellen sollten auf einer einzigen Festplatte gespeichert werden. Die Umverteilung der Tabellenbereiche auf zwei Festplatten auf dem zweiten Rechner, führte wegen unterschiedlicher Drehgeschwindigkeiten der Festplatten (4000 U/Min und 7200 U/Min) zu nicht vergleichbaren Ergebnissen. Für manche Konfigurationsparameterwerte, war es nicht auszumachen, ob der Performanceeinbruch auf die langsamere Festplatte oder den Parameterwert zurückzuführen ist.

Bei genauerer Betrachtung der Ergebnisse aus früheren Testläufen wurde festgestellt, dass RDF-S3 die meiste Zeit mit dem Speichern von RDF-Properties des VRP-Modells, das beim Parsen erzeugt wird, beschäftigt war. Dies ist nicht überraschend, wenn man bedenkt, dass dafür auf verschiedene Tabellen zugegriffen werden muss und u.U. auch einige zusätzliche Tabellen erzeugt werden müssen. Die Tabelle RESOURCES hingegen war die am stärksten frequentierte Tabelle. Daher war es wichtig, diese beiden Tabellen in eigenen Tabellenbereichen unterzubringen. Vor allem höhere Werte für *extentsize* und *prefetchsize* sollen die Performance steigern. Da die DMS-Tabellenbereiche in RAW-Behältern auch die bis dahin besten Ergebnisse lieferten, wurden die beiden Tabellenbereiche PROPRTS\_TB für die PROPERTIES-Tabelle und RESRCS\_TB für die RESOURCES-Tabelle in eigene RAW-Behälter gespeichert. Die dazugehörigen Indizes für die beiden Tabellen, wurden wegen einfacherer Verwaltung und besserer Performance in die entsprechenden Tabellenbereiche mitgespeichert. An dieser Stelle sei jedoch gesagt, dass die RAW-Behälter aus Mangel an Festplatten nur als unformatierte Partitionen einer Festplatte und nicht ganze unformatierte Laufwerke sind.

Wie die Abb. 4.10 zeigt, war bei den Tests mit großen Dateien die Trennung der Tabellen PROPERTIES und RESOURCES in eigene Tabellenbereiche sogar mit einem Performanceverlust verbunden, da der Systemaufwand (OVERHEAD<sup>11</sup>) für den Tabellenbereichwechsel und damit Behälterwechsel, insbesondere beim Einsatz der Festplatte mit

---

<sup>11</sup>Der OVERHEAD ist ein Schätzwert in Millisekunden für die Zeit, die der Behälter benötigt, bevor irgendwelche Daten in den Speicher gelesen werden. In diesen Wert fließen der Aufwand für den E/A-Controller des Behälters und die Latenzzeit der Platte, zu der auch die Suchzeit der Platte gehört, mit ein.

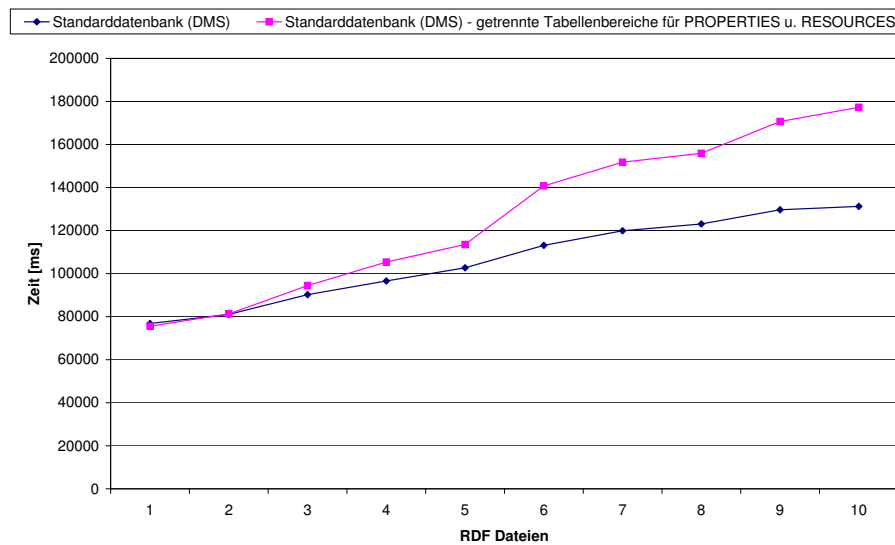


Abbildung 4.10: Trennung der Tabellen PROPERTIES und RESOURCES.

4000 U/Min, den Geschwindigkeitsgewinn übersteigt. Es ist nicht realistisch, eine Performancesteigerung schon durch die Verteilung der Daten auf einer einzigen Festplatte zu erwarten. Stehen jedoch mehrere Laufwerke mit mehreren Platten-Controllern zur Verfügung, sollten die Behälter auf getrennten Laufwerken angelegt werden.

Die Trennung von Tabellen in eigene Tabellenbereiche sollte außerdem die Erzeugung zusätzlicher Pufferpools einhergehen. Im nächsten Abschnitt wird die Optimierung von Pufferpools vorgenommen. Auch bei den Tests mit kleinen und mittelgroßen Dateien konnte der durch die Trennung der Tabellen in mehrere Tabellenbereiche entstehende Performanceverlust beobachtet werden.

## 4.6 Der Pufferpool

Der Pufferpool<sup>12</sup> ist ein wichtiger Bestandteil des Datenbankverwaltungssystems und beeinflusst die Performance wesentlich, da die Daten aus dem Pufferpool dem zuständigen Agenten viel schneller zur Verfügung stehen, als wenn sie aus der Festplatte geholt werden müssen.

Bei der Optimierung des Pufferpools gibt es zwei Möglichkeiten. Die erste Möglichkeit besteht darin, die Größe des Pufferpools zu verändern. Die zweite Möglichkeit wäre, zusätzliche Pufferpools zu erzeugen. Für beide Methoden muss aber genug Arbeitsspeicher im Server zur Verfügung stehen. Sowohl der Server als auch der Client müssten sich die vorhandenen 512 MB RAM aufteilen. Daher war die Möglichkeit einer großzügigen Gestaltung der Pufferpools auszuschließen. Auch die Anzahl der Pufferpools war nicht ohne weiteres

<sup>12</sup>Siehe Abschnitt 2.3.1 auf S.19

zu erhöhen. Es erschien uns angemessen, drei Pufferpools zu erzeugen: je einen Pufferpool für die Tabellenbereiche PROPRTS\_TB und RESRCS\_TB und einen für die restlichen Tabellenbereiche. Die Standardgröße für Pufferpool beträgt 250 Seiten. Mögliche Werte für die Pufferpoolgröße dagegen sind zwischen 2 und 524228 Seiten.

Es ist wichtig, dass mit der Vergrößerung des Pufferpools auch an eine gleichzeitige Vergrößerung des DatenbankzwischenSpeichers (*dbheap*-Konfigurationsparameter – Kap.2 Abb. 2.9) gedacht wird. Für jede Datenbank gibt es einen DatenbankzwischenSpeicher, der vom Datenbankmanager für alle Anwendungen, die auf diese Datenbank zugreifen, verwendet wird. Er enthält Steuerblockdaten für Tabellen, Indizes, Tabellenbereiche und Pufferpools.

Wie bereits dargelegt, wurde die Trennung von Indizes und eigentlichen Daten in mehrere Pufferpools aus Knappheit an Arbeitsspeicher nicht durchgeführt, obwohl dies empfohlen wird. Die hohe Anzahl der Benutzertabellen bzw. Indizes sprach zusätzlich für diese Entscheidung.

Bei der Überwachung der Pufferpoolgröße mit dem Snapshot Monitor, ist es sehr hilfreich die so genannte *Hit Ratio* Variable zu berechnen. Sie kann mit der Formel

$$HitRatio = 100 * (1 - \frac{physLiP+physLiPI}{logLiP+logiPI})$$

wobei:

- physLiP = physische Lesevorgänge im Pufferpool
- physLiPI = physische Lesevorgänge im Pufferpoolindexseiten
- logLiP = logische Lesevorgänge im Pufferpool
- logLiPI = logische Lesevorgänge im Pufferpoolindexseiten

berechnet werden und bezeichnet den Prozentsatz von erfolgreichen Zugriffen auf die Pufferpoolseiten. Wenn der Wert von Hit Ratio bei der Datenbank im laufenden Betrieb unter eine selbst definierte Grenze fällt – empfohlen wird ein Wert von 90 % – ist ein zu kleiner Pufferpool gewählt worden. Je nach verfügbarem Arbeitsspeicher sollte der Pufferpool dann vergrößert werden.

Zur Performancesteigerung kann auch ein zusätzlicher Pufferpool für die Systemkatalogtabellen einen wichtigen Beitrag leisten, besonders wenn die Systemkatalogtabellen einen eigenen Tabellenbereich haben. Die Systemkatalogtabellen werden vom Datenbankmanager sehr häufig benutzt.

Während der Tests mit großen Dateien und dem Standardpufferpool von 250 Seiten, fiel auf, dass der *Hit Ratio* nur einen Wert von 16 % erreichte<sup>13</sup>. In diesem Fall war der Standardpufferpool so klein, dass sehr viele Seiten zuerst ausgelagert und dann neue Seiten

---

<sup>13</sup>Dies war darauf zurückzuführen, dass die Protokolldateien nicht ausreichend groß waren. Siehe Abschnitt Logging.

geholt werden mussten, um überhaupt wenige Treffer erzielen zu können. Bei den Tests mit mittelgroßen Dateien war der Standardpufferpool zwar auch klein, aber er zeigte dennoch bessere Trefferquoten als bei den Tests mit großen Dateien (Abb. 4.11). Da die kleinen Dateien nicht so viele Daten enthalten, reichten die vorgegebenen 250 Seiten fast aus, um die Daten permanent im Pufferpool halten zu können (rote Linie - Abb. 4.12). Dennoch waren zwei zusätzliche Pufferpools von je 250 Seiten notwendig, je einer für die Tabellenbereiche PROPRTS\_TB und RESRCS\_TB, um die Performance deutlich zu steigern (blaue Linie - Abb. 4.12). Der Wert von Hit Ratio stieg dabei über 99 %.

```

DB2 CLP - db2setcp.bat DB2SETCP.BAT DB2.EXE
Gesamtanzahl Sortiervorgänge           = 0
Gesamte Sortierzeit <ms>              = Nicht gesamme
It
Überläufe bei Sortierung              = 0
Aktive Sortiervorgänge                 = 0

Logische Lesevorgänge im Pufferpool    = 525923
Physische Lesevorgänge im Pufferpool   = 378709
Asynchrone Lesevorgänge in Poolseiten = 133655
Schreibvorgänge im Pufferpool         = 2804
Asynchrone Schreibvorgänge in Poolseiten = 2415
Logische Lesevorgänge im Pufferpoolindex = 845698
Physische Lesevorgänge im Pufferpoolindex = 2802
Asynchrones Lesen der Poolindexseiten = 12
Schreibvorgänge im Pufferpoolindex    = 2384
Asynchrone Schreibvorgänge in Poolindexseiten = 2129
Gesamtzeit der Lesevorgänge im Pufferpool <ms> = 5528
Gesamtzeit der Schreibvorgänge im Pufferpool <ms> = 25965
Gesamtzeit für asynchrone Lesevorgänge = 1256
Gesamtzeit für asynchrone Schreibvorgänge = 25532
Asynchrone Leseanforderungen          = 24428
Warteschlange bei LSM-Löcke           = 70
Warteschlange bei Mangel an Leeren Seiten = 449
Warteschlange bei Höchstanzahl benutzer Seiten = 263
Wartezeit für Writablesezugriff <ms> = 147
  
```

Abbildung 4.11: Screenshot einer Momentaufnahme während eines Tests mit mittelgroßen Dateien. Aus den Angaben ergibt sich ein Wert von Hit Ratio = 72 %.

#### 4.6.1 Direkte Veränderungen am Pufferpool

In der ersten Testreihe wurde untersucht, ob und wieviel Performancesteigerung erzielt werden kann, wenn für verschiedene Tabellenbereiche getrennte Pufferpools mit der Standardpufferpoolgröße von 250 Seiten verwendet werden. Dabei blieb der Wert von *dbheap* zuerst unverändert auf vorgegebene 300 Seiten.

In der zweiten Testreihe wurde untersucht, in welchem Maß die Vergrößerung des Pufferpools auf 2500 bzw. 7500 Seiten auf die Performance wirkt. Dabei wurde die Anzahl der Pufferpools nicht erhöht.

Bei den Tests mit kleinen Dateien bewirkte die Erhöhung der Pufferpoolanzahl auf 3 (mit je 250 Seiten) eine deutliche Performancesteigerung (blaue Linie - Abb. 4.12). Mit der Erhöhung der Anzahl der Pufferpools verschwanden außerdem auch die hohen Schwankungen zwischen einzelnen Zeiten. Das ist darauf zurückzuführen, dass die angeforderten Seiten fast immer im Pufferpool waren und die zuständigen Datenbankagenten nicht auf Daten warten mussten. In einer zusätzlichen Testreihe wurde der Versuch gemacht, die Größe

aller drei Pufferpools von 250 Seiten auf 2500 Seiten zu erhöhen (grüne Linie - Abb. 4.12). Die Testergebnisse stimmten mit denen der Testreihen mit drei Pufferpools mit je 250 Seiten überein. Auch der Versuch aus der zweiten Testreihe, den einzigen Pufferpool von 250 Seiten auf 2500 Seiten zu vergrößern, brachte keine weitere Verbesserung.

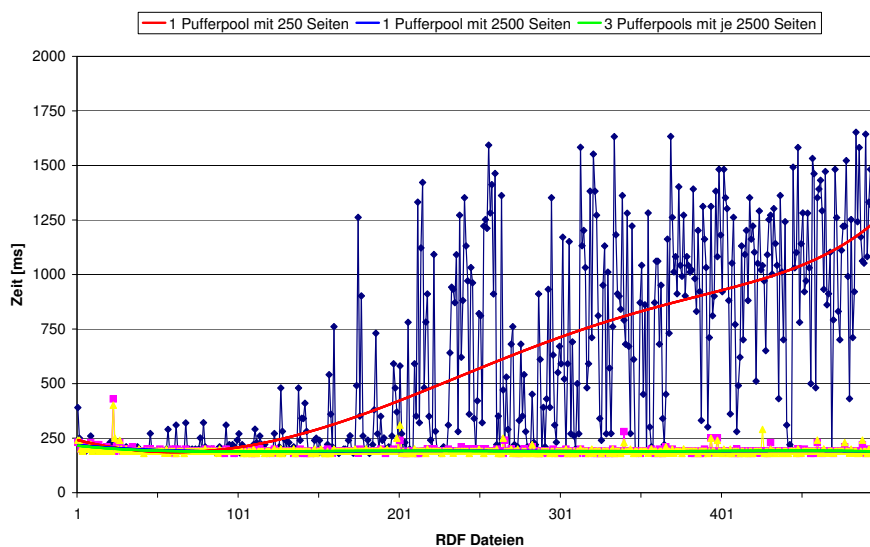


Abbildung 4.12: Datenbankkonfiguration mit unterschiedlicher Pufferpoolanzahl und -Größe. – kleine Dateien.

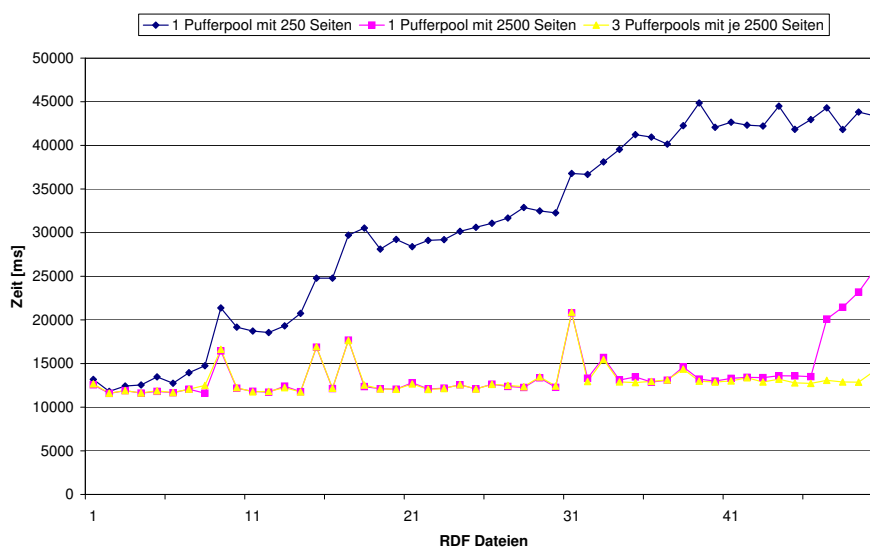


Abbildung 4.13: Datenbankkonfiguration mit unterschiedlicher Pufferpoolanzahl und -Größe. – mittelgroße Dateien.

Bei den Tests mit mittelgroßen Dateien ergab sich ein anderes Bild. Die Erhöhung

der Anzahl der Pufferpools auf drei reichte hier für eine deutliche Performancesteigerung nicht aus, da die 250 Seiten für einen Pufferpool dennoch zu knapp bemessen waren. Die Vergrößerung des einzigen Pufferpools von 250 auf 2500 Seiten brachte zwar eine deutliche Performancesteigerung, aber am Ende erhöhten sich die Zeiten wieder, was darauf zurückzuführen ist, dass auch die 2500 Seiten nicht völlig ausreichend waren (Abb. 4.13 - lila Linie). Erst eine Erhöhung der Pufferpoolanzahl auf drei mit je 2500 Seiten zeigte eine deutliche und dauerhafte Leistungssteigerung (Abb. 4.13 - gelbe Linie). Der Versuch mit noch mehr Pufferpools brachte jedoch keine weiteren messbaren Verbesserungen.

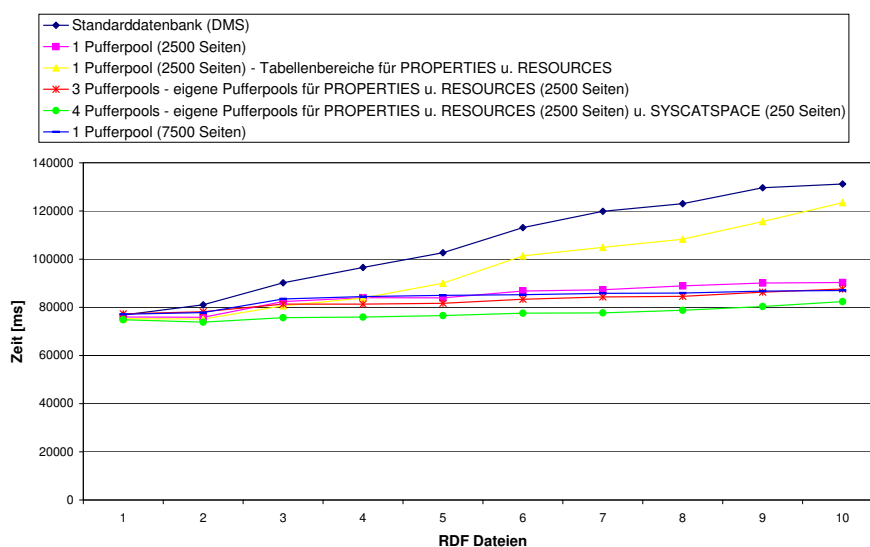


Abbildung 4.14: Datenbankkonfiguration mit unterschiedlicher Pufferpoolanzahl und -Größe. – große Dateien.

Die Tests mit großen Dateien zeigten, dass allein die Vergrößerung des Pufferpools einen deutlichen Performancegewinn bewirkt (lila Linie - Abb. 4.14), wenn keine Trennung der Tabellen PROPERTIES und RESOURCES vorgenommen wird. Im anderen Fall ist mit einem Performanceverlust zu rechnen (gelbe Linie). Dies ist darauf zurückzuführen, dass sich alle Tabellenbereiche in einer einzigen Festplatte befanden. Nach [DB2-02, S.22] soll bei der Verwendung der DMS-Tabellenbereiche auch die Möglichkeit jeden Behälter einer anderen Platte zuzuordnen, in Betracht gezogen werden. Dadurch wird die Tabellenkapazität vergrößert und die Möglichkeit geschaffen, parallele E/A-Operationen zu nutzen.

Um in Bezug auf die Pufferpoolgröße eine Gleichheit zu schaffen, wurde eine Testreihe mit einer Pufferpoolgröße von 7500 Seiten durchgeführt (blaue Linie). Dabei wurde keine Trennung der Tabellen vorgenommen. Die Tests zeigten, dass die Performance fast genauso gut war wie bei den Tests mit drei Pufferpools von je 2500 Seiten und getrennten Tabellen (rote Linie). Die Trennung der Tabellen in verschiedene Behälter konnte die Möglichkeit der parallelen E/A-Operationen nicht nutzen.

Einzig die Erzeugung eines zusätzlichen Pufferpools von 250 Seiten für die Systemkatalogtabellen (grüne Linie) zeigte eine deutliche bessere Performance.

Auch in diesem Fall ist es so, dass erst im Verlauf des Tests die Performanceunterschiede deutlicher werden, da die Datenbank mit immer mehr Daten gefüllt wird.

### 4.6.2 Die Veränderung an den abhängigen Parametern

Wie bereits erwähnt, bewirkt die Erhöhung der Anzahl von Pufferpools und/oder die Vergrößerung von Pufferpools allein u. U. noch nicht die erhoffte Verbesserung der Performance, wenn der Datenbankzwischenpeicher (*dbheap*)<sup>14</sup> als ein wichtiger Einflussfaktor, nicht mit abgestimmt wird. Ist die Größe des Datenbankzwischenpeichers zu klein gewählt worden, kann es zu einem Ressourcenverwaltungsproblem kommen und die Datenbank kann nicht so viele Abfragen bearbeiten, wie es eigentlich möglich wäre.

Bei allen Tests dieser Art zeigte sich entgegen allen Erwartungen keine Verbesserung der Performance. Dies ist wohl darauf zurückzuführen, dass die Optimierung der Konfigurationsparameter durch die verwendete Hardware gebremst wird und an ihre Grenzen stößt.

## 4.7 Das Logging

Der Logger hat die Aufgabe, sämtliche Vorgänge in der Datenbank auf ein externes Medium wie z.B. die Festplatte zu speichern. Es ist wichtig, dass die physische Platzierung der Protokolldateien (Log file) und deren Anzahl schnelleres Schreiben erlauben. Für die Wiederherstellung der Datenbank nach einem Ausfall ist dagegen die Lesegeschwindigkeit wichtiger. Da RDF-S3 hauptsächlich das Speichern der Daten zur Aufgabe hat, war das schnellere Schreiben als wichtiger eingestuft. Zusätzlich sollten die Protokolldateien möglichst auf andere Laufwerke verteilt werden und aus Sicherheitsgründen sollten sie auch gespiegelt werden.

Bei der Durchführung der Tests mit großen Dateien fiel nach Wiederholung einiger Testreihen auf, dass es auf der Festplatte keinen freien Speicher mehr gab. Nach detaillierter Untersuchung der Ursache dafür wurde festgestellt, dass die Datei *db2diag.log*, in der alle (Fehler)Diagnosen (*diaglevel*-Parameter<sup>15</sup>) aufgezeichnet werden, eine Größe von über 7 GB erreicht hatte. Da bei den Tests mit großen Dateien insgesamt ca. 1.200.000 INSERT-, UPDATE-Operationen in einer Testreihe ausgeführt wurden, gab es dementsprechend sehr viele Sperren. Der Speicherplatz, der für das Sperren aufgewendet wird, wird durch den Konfigurationsparameter *locklist* (Sperrenliste) der Datenbank gesteuert. Der Datenbankmanager zeichnet für jede Datenbank in der Sperrenliste alle gehaltenen

---

<sup>14</sup>Wertebereich für *dbheap*-Parameter beträgt 32 bis 524288 Seiten.

<sup>15</sup>Dieser Parameter legt die Art der Diagnosefehler fest, die in der Fehlerprotokolldatei eingetragen werden sollen.

Sperren auf. Der Standardwert von 25 Seiten<sup>16</sup> für die Sperrenliste war für diese Tests zu klein. Im günstigsten Fall können in der Sperrenliste ca. 2800 Sperren aufgezeichnet werden. In den Tests mit großen Dateien zeigten die Momentaufnahmen, dass es zeitweise ca. 20000 Sperren<sup>17</sup> gab. Wenn sich die Sperrenliste füllt, kann sich die Leistung aufgrund von Sperreneskalationen und verringertem gemeinsamen Zugriff auf gemeinsam verwendete Objekte in der Datenbank verschlechtern. Alle diese Ereignisse werden als Warnungen in der *db2diag.log* Datei aufgezeichnet. Da die Sperrenliste sich sehr schnell füllte, musste der Datenbankmanager sehr viele Sperreneskalationen durchführen und zugleich alle Warnungen aufzeichnen. Damit konkurrierten die Protokolldateien und die Datenbankbehälter um die Bewegung derselben Plattenschreib-/leseköpfe, was auf die Gesamtlaufzeit in sehr hohem Maße negativ auswirkte (Abb. 4.15). Nach einer einzigen Testreihe mit großen Dateien entstand eine über 500 MB große *db2diag.log* Datei. Um dies zu verhindern, wurde der Parameterwert von *diaglevel* von 3 (Aufzeichnung aller Fehler und Warnungen) auf 1 gesetzt (Aufzeichnung nur kritischer Fehler). Durch diese Maßnahme war das eigentliche Problem jedoch noch nicht gelöst, denn das Auftreten von sehr vielen Sperren und Sperreneskalationen bestand weiter.

Nach weiteren detaillierten Untersuchungen, wurde schließlich die eigentliche Ursache des aufgetretenen Problems ausgemacht – der mit 250 Seiten zu klein eingestellter Standardparameterwert für die Protokolldateien (*logfilesize*). Eine primäre Protokolldatei kann wieder verwendet werden, wenn die in ihr aufgezeichneten Änderungen festgeschrieben wurden. Wenn die Größe der Protokolldatei beschränkt ist und Anwendungen eine große Anzahl von Änderungen an der Datenbank vorgenommen haben, ohne die Änderungen festzuschreiben, kann eine primäre Protokolldatei schnell voll werden. Dadurch wurde die oben beschriebene Kettenreaktion ausgelöst und es kam zum Performanceeinbruch. Damit das Problem nicht mehr auftreten konnte, wurde die Protokolldateigröße auf 2500 Seiten gesetzt und die Anzahl der primären bzw. sekundären Protokolldateien auf 5 bzw. 3 erhöht. Das hier geschilderte Problem trat bei den Tests mit kleinen und mittelgroßen Dateien nicht auf. Die neuen Tests mit großen Dateien zeigten sogar eine Performanceverbesserung um ca. 90 % (Abb. 4.15). Somit verkürzte sich die Ausführungsdauer einer Testreihe mit großen Dateien von ca. 2 Stunden und 50 Minuten auf ca. 35 Minuten.

Der Konfigurationsparameter *lrgbfsz* gibt die Menge des DatenbankzwischenSpeichers (*dbheap*) an, der als Puffer für Protokollsätze verwendet wird, bevor diese auf den Datenträger geschrieben werden. Durch Puffern der Protokollsätze werden die E/A-Operationen für die Protokolldateien effektiver, da weniger Schreiboperationen ausgeführt werden müssen und bei jeder Schreiboperation mehrere Protokollsätze gleichzeitig auf die Festplatte geschrieben werden. Der Standardwert von *lrgbfsz* ist dabei mit 8 Seiten angegeben und ist ein wichtiger Parameter bei der Leistungsoptimierung.

---

<sup>16</sup>Auf 32-Bit-Plattformen erfordert jede Sperre 36 oder 72 Byte der Sperrenliste – je nachdem, ob für das Objekt andere Sperren aktiv sind.

<sup>17</sup>Mit angepasstem Parameterwert für *locklist*.

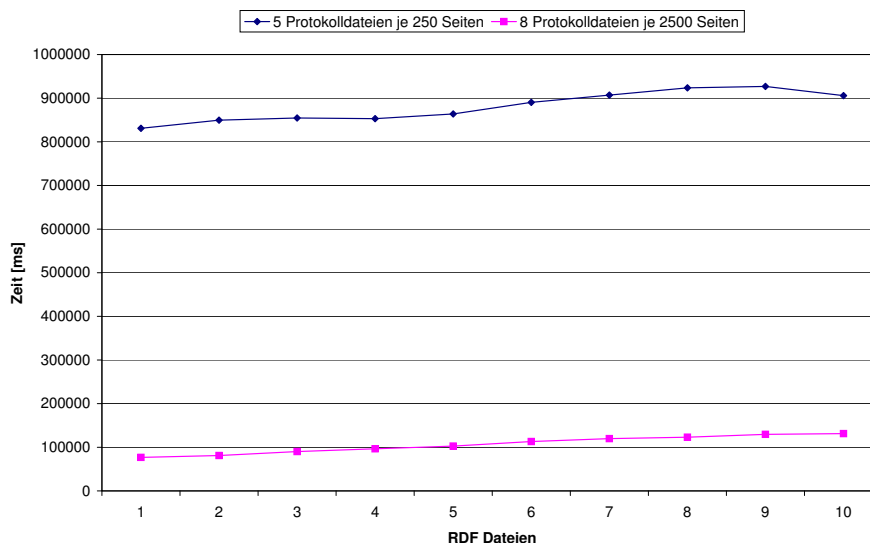


Abbildung 4.15: Optimierung der Protokolldateianzahl und -Größe.

Nur in einem einzigen Test mit großen Dateien, der lediglich einmal durchgeführt wurde, wurde der Wert von *luginfsz* von 8 auf 80 erhöht.

Aus diesem Test ging nicht eindeutig hervor, ob eine Verbesserung der Performance eingetreten war, da sich die Zeiten im Testverlauf zweimal änderten. Am Anfang waren die Zeiten schlechter, um anschließend eine bessere Performance zu zeigen und am Ende dann gleich zu ziehen. Dies könnten systembedingte Schwankungen gewesen sein. Es ist aber wohl aufgrund der Erhöhung dieses Parameterwertes mit keiner Verbesserung zu rechnen. Die Erhöhung des *locklist*-Parameters hingegen, führte zu einer deutlichen Verschlechterung der Performance (Abb. 4.16 - gelbe Linie).

## 4.8 Die Parallelisierung

DB2 unterstützt parallele Umgebungen in erster Linie auf symmetrischen Multiprozessormaschinen (SMP-Maschinen) und nur in begrenztem Maß auch auf Einzelprozessormaschinen. Die Parallelisierung (Konfigurationsparameter *intra\_parallel*) von Anfragen ist ein sehr wichtiger Einflussfaktor bei der Performanceoptimierung.

Die in den hier dokumentierten Testreihen verwendete Hardware erlaubt nur die Parallelität innerhalb von Festplattenpartitionen. In [DB2-00, S.199] wird die Verwendung dieser Art von Parallelität insbesondere für die OLAP-Belastungen empfohlen. Zudem wird darauf hingewiesen, dass die Erhöhung des Parallelitätsgrades bei Einzelprozessormaschinen den Hauptspeicherbedarf und die CPU-Belastung im System erhöht. Dies war wohl der Grund, weshalb bei den Tests eine minimale Performanceverschlechterung auftrat. Au-

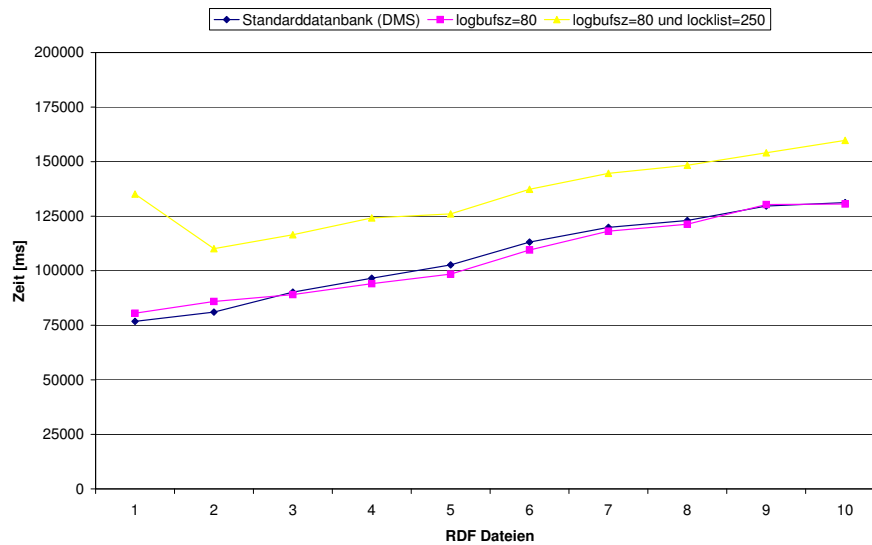


Abbildung 4.16: Auswirkungen eines erhöhten Wertes für *logbufsiz*-Parameter.

ßerdem profitieren die Abfragen, die eine Ausführungszeit unter einer Sekunde haben, im Allgemeinen nicht von der Parallelisierung. Der Systemaufwand für den Prozesswechsel übersteigt den möglichen Geschwindigkeitsgewinn.

## 4.9 Konfiguration anderer Parameter

Eine wichtige Rolle bei der Optimierung der Konfigurationsparameter können die Parameter *num\_iocleaners* und *num\_ioservers* spielen.

Mit dem Parameter *num\_iocleaners* kann die Anzahl asynchroner Seitenlöschfunktionen für eine Datenbank angegeben werden. Diese Seitenlöschfunktionen schreiben geänderte Seiten aus dem Pufferpool auf die Festplatte, bevor der Bereich im Pufferpool von einem Datenbankagenten angefordert wird. Daher müssen Datenbankagenten in der Regel nicht die Auslagerung geänderter Seiten abwarten, bevor sie den Speicherbereich im Pufferpool nutzen können. In [DB2-02] wird empfohlen den Wert dieses Parameters zwischen 1 und der Anzahl der physischen Speichereinheiten, die für diese Datenbank verwendet werden, zu setzen. Es wird außerdem empfohlen, den Wert zu erhöhen, wenn eine der folgenden Bedingungen erfüllt ist:

- „Schreibvorgänge im Pufferpool“ ist viel größer als „asynchrone Schreibvorgänge in Poolseiten“.
- „Schreibvorgänge im Pufferpoolindex“ ist viel größer als „asynchrone Schreibvorgänge in Poolindexseiten“.

Eng mit der Anzahl physischer Festplatten zusammenhängend ist die Einstellung des Wertes des *num\_ioservers*-Parameters. Mit diesem Parameter wird die Anzahl der E/A-Server für eine Datenbank definiert. E/A-Server werden für Datenbankagenten verwendet, um Vorablese-E/A-Operationen und asynchrone E/A-Operationen von Dienstprogrammen wie BACKUP (Sicherung) und RESTORE (Wiederherstellung) auszuführen. Auch hier wird ein Wert empfohlen der um 1 oder 2 höher ist als die Anzahl der physischen Einheiten, auf denen sich die Datenbank befindet.

Bei der Durchführung der Tests zeigte die Überwachung der Verhältnisse von Schreibvorgängen und asynchronen Schreibvorgängen für die Pufferpool(index)seiten, dass die Erhöhung der Anzahl asynchroner Seitenlöschfunktionen nicht notwendig war. Dementsprechend brachte die Erhöhung des Parameterwertes auf 3 Seitenlöschfunktionen auch keinen Performancegewinn.

Die Erhöhung der Anzahl von E/A-Servern hingegen war sogar mit einem kleinen Performanceverlust verknüpft. Der Grund dafür ist darin zu finden, dass die zusätzlichen E/A-Server bei Benutzung einer Festplatte nicht benötigt wurden. Da die E/A-Server auch im inaktiven Zustand einen geringfügigen Systemaufwand verursachen, kam es zu dem festgestellten Performanceverlust.

## 4.10 Die optimierte Datenbank

Bei der Erstellung der optimalen Konfiguration wurden die in diesem Kapitel festgestellten positiven Veränderungen kombiniert und einem erneuten Test unterzogen. Um realitätsnahe Testergebnisse zu bekommen, wurden diese Tests mit der Methode des Tests einzelner Dateien durchgeführt.

Mit Ausnahme von temporären Systemtabellen wurden alle anderen Tabellen (Benutzer- und Systemtabellen) in DMS-Tabellenbereiche angelegt. Die DMS-Tabellenbereiche wurden als Datei-Behälter deklariert. Da die am meisten belastete Tabelle die RESOURCES Tabelle war, wurde sie in einem eigenen Tabellenbereich untergebracht – RESRCS\_TB. Die Größe dieses Tabellenbereiches wurde auf 250 MB festgelegt. Auf die Trennung der Tabelle PROPERTIES in einem eigenen Tabellenbereich wurde hier verzichtet. Zum einem wäre die Trennung der Tabelle mit zusätzlichem Aufwand verbunden gewesen und zum anderen bedeutete der Verzicht auf die Trennung keinen Performanceverlust. Die Trennung anderer Tabellen in eigene getrennte Tabellenbereiche kann in Systemen mit mehreren physischen Festplatten dagegen durchaus zu einer besseren Performance führen. Diese Tests konnten auf der Basis der für diese Diplomarbeit verwendeten Hardware nicht getestet werden. Alle übrigen Benutzertabellen wurden in einem DMS-Tabellenbereich (USERSPACE1) als Datei-Behälter mit einer Größe von 200 MB untergebracht. Die Systemtabellen wurden in einem DMS-Tabellenbereich als Datei-Behälter mit einer Größe von 100 MB angelegt. Da die Systemtabellen vom Datenbankmanager sehr häufig verwendet werden, wird dadurch eine Performancesteigerung erreicht.

Da die RESOURCES Tabelle eine sehr starke frequentierte Tabelle war, wurde der Wert

für die Parameter *extentsize* und *prefetchsize* dem starken Zugriff entsprechend auf 64 festgelegt.

Bei den hier vorgestellten Tests war die Anzahl der Pufferpools und deren Größe ein wichtiger Faktor für die Performancesteigerung. Daher wurde die Anzahl der Pufferpools auf Drei festgelegt. Ein Pufferpool mit einer Größe von 250 Seiten wurde für die Systemkatalogtabellen angelegt. Dadurch war die bessere Abtrennung der Systemkatalogtabellen möglich. Aus demselben Grund wurde auch für den RESRCS\_TB Tabellenbereich ein eigener Pufferpool mit 3500 Seiten angelegt. Für alle anderen Tabellen wurde ein dritter Pufferpool mit 3500 Seiten festgelegt.

Zusätzlich wurden auch einige andere Parameter verändert, die in der folgenden Tabelle aufgelistet werden.

Parameter	Parameterwert (optimiert)	Standardwert
<i>logfil_siz</i>	2500 Seiten	250 Seiten
<i>logprimary</i>	5	3
<i>logsecond</i>	3	2
<i>dbheap</i>	1000 Seiten	300 Seiten
<i>applheapsz</i>	500 Seiten	256 Seiten
<i>pckcache_sz</i>	1000 Seiten	320 Seiten = $8 * maxappls$

Tabelle 1. Einige Konfigurationsparameterwerte für die optimierte Datenbank.

Eine andere Testreihe zeigte, dass auch ohne Trennung der Tabelle RESOURCES kein Performanceverlust hingenommen werden musste. Entscheidend war dagegen die Erhöhung der Gesamtgröße des Pufferpools auf 7500 Seiten. Deshalb wurde auch für die abschließenden Tests keine Trennung der Tabellen vorgenommen.

Die folgende Abbildung zeigt die gemessenen Laufzeiten bei den Tests mit kleinen Dateien im Vergleich zu den Laufzeiten der Standardkonfiguration.

Man erkennt in der Abbildung, dass die optimierte Konfiguration eine deutlich bessere Performance aufweist – im Durchschnitt ca. 75 %. Insbesondere die Optimierung der Pufferpools sowie die Erzeugung der DMS-Tabellenbereiche waren ausschlaggebend für den Performancegewinn.

Auch in der folgenden Abbildung – dem Vergleich der Standardkonfiguration und der optimierten Konfiguration mit Dateien mittlerer Größe – wird deutlich, wie Hoch der Performancegewinn ist, wenn die Tabellenbereiche als DMS deklariert werden und die Pufferpools ausreichend groß sind. Bei den mittelgroßen Dateien betrug der Performancegewinn ca. 71 % im Durchschnitt.

Bei den Tests mit kleinen Dateien und Dateien mittlerer Größe sind mit der Optimierung der Konfigurationsparameter die Schwankungen zwischen einzelnen Zeiten auf einem sehr niedrigen Niveau gesunken. Die bessere Verwaltung der Tabellenbereiche (DMS) durch DB2 und der große Pufferpool, in dem die meisten angeforderten Daten zu finden waren, sind als Hauptgrund zu nennen.

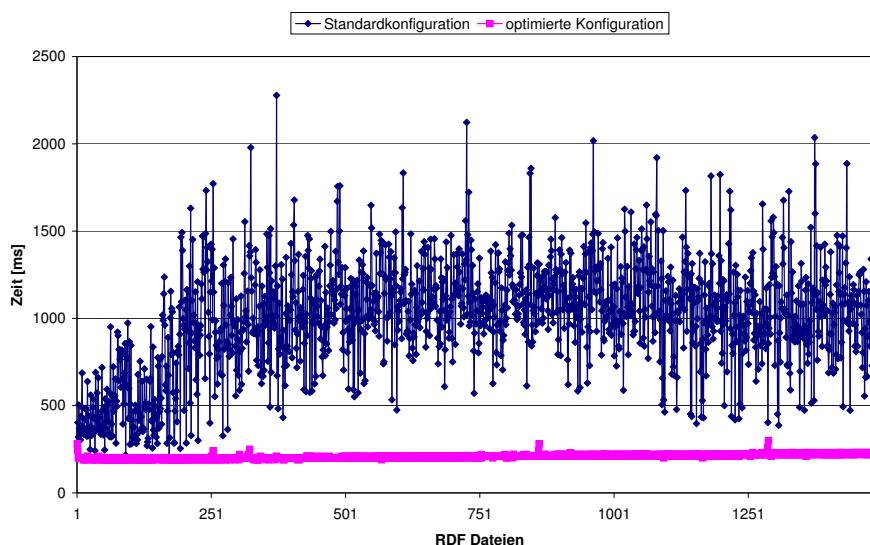


Abbildung 4.17: Standardkonfiguration vs. optimierte Konfiguration – kleine Dateien.

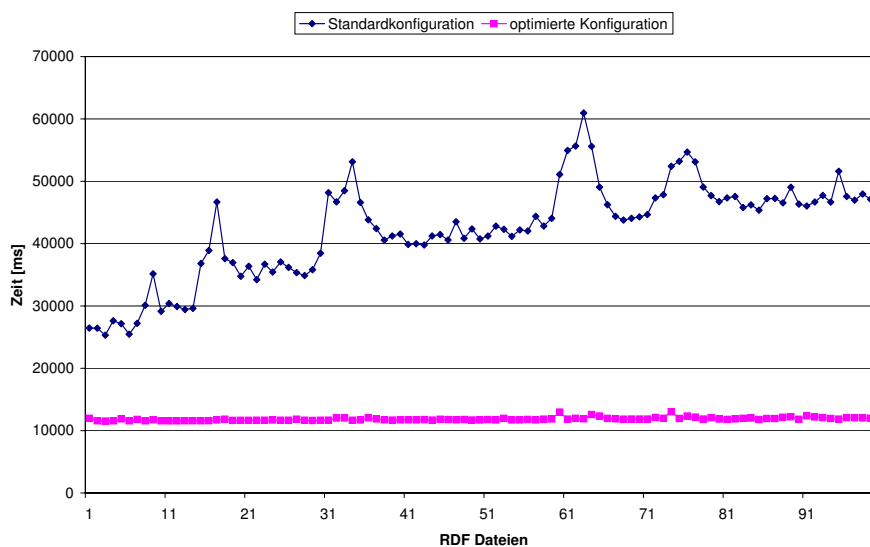


Abbildung 4.18: Standardkonfiguration vs. optimierte Konfiguration – mittelgroße Dateien.

Auch bei den Tests mit großen Dateien kam es zu einer deutlichen Performancesteigerung. Um die Tests mit großen Dateien durchzuführen – Methode des Tests einzelner Dateien – war es notwendig eine modifizierte Version von RDF-S3 (1.8Pre) zu verwenden. Dabei wurde die Batch-Abarbeitung der POI-Einträge auf max. 500 begrenzt, da es zu einem **Batch.Update.Exeption** in DB2 kam. Dies führte dazu, dass DB2 die Batch-Abarbeitung der POI-Einträge verworfen hat und somit die Laufzeiten verfälscht wurden.

Die folgende Abbildung zeigt die Tests mit großen Dateien. Dabei wurde bei der Stan-

Standardkonfiguration die Größe der Protokolldatei von 250 Seiten auf 2500 Seiten sowie die Anzahl der primären und sekundären Protokolldateien auf 5 bzw. auf 3 erhöht, da sonst die Durchführung der Testreihe aus Mangel an Platz im Transaktionsprotokoll für die Datenbank nicht möglich gewesen wäre.

Das Ausmaß des Performancegewinns der optimierten Konfiguration von ca. 71 %

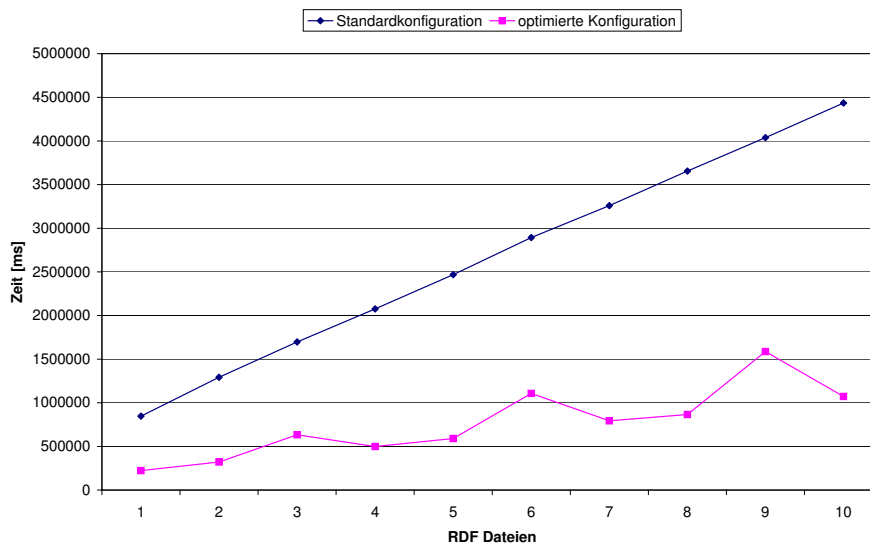


Abbildung 4.19: Standardkonfiguration vs. optimierte Konfiguration – große Dateien.

im Durchschnitt gegenüber der Standardkonfiguration überraschte. Der Performancegewinn ist vor allem durch die Deklarierung von Tabellenbereichen als DMS und durch den großen Pufferpool erzielt worden.

Wie in diesem Abschnitt dargestellt, ließ sich bei allen Testreihen durch wenig Aufwand ein enormer Performancegewinn erzielen, der in allen drei Testreihen über 70 % betrug. Dieses Ergebnis lag weit über dem, was am Anfang dieser Diplomarbeit erhofft war. Daher kann nur empfohlen werden, in jedem Falle eine Umstellung der Standardkonfiguration vorzunehmen. Dabei sollten die Bemühungen insbesondere auf die Umstellung der Tabellenbereiche in DMS und die Anpassung der Pufferpoolgröße konzentriert werden. Die zusätzliche Trennung der Tabellen – hier ist die Tabelle RESOURCES wegen ihrer starken Benutzung hervorzuheben – kann bei Systemen mit mehreren Festplatten eine weitere Performancesteigerung bewirken.

## 4.11 Schema-Daten

Die Tests mit Schema-Daten (Datei *culture.rdf*) wurden für die Standarddatenbankkonfiguration und die Konfigurationswerte der optimierten Datenbank aus dem vorherigen

Abschnitt durchgeführt. Auch hier gibt es eine deutliche Performancesteigerung mit den optimierten Konfigurationsparametern (Abb. 4.20). Allerdings war hier der Performancegewinn mit ca. 51 % nicht so stark wie bei den Tests mit RDF-Daten. Dies ist darauf zurückzuführen, dass die Zeiten insgesamt auf sehr niedrigem Niveau waren, da die Erzeugung der Tabellen keinen großen Zeitaufwand erforderten. Die Schwankungen einzelner Zeiten bei der Standardkonfiguration sind systembedingt, da die Tabellenbereiche als SMS definiert waren und somit von anderen Prozessaktivitäten beeinflusst werden.

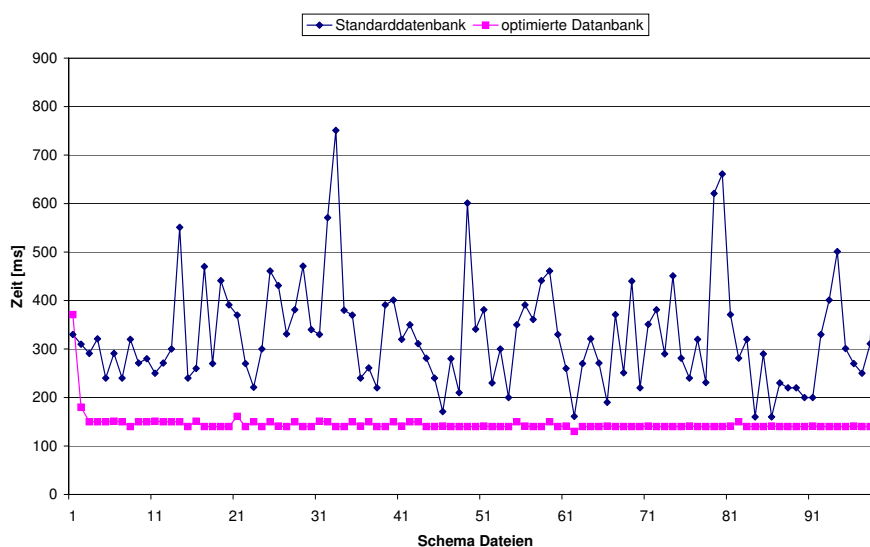


Abbildung 4.20: Schema-Daten – Standardkonfiguration vs. optimierte Konfiguration.

## 4.12 Stream Based Parsing

Wie bereits in Kapitel 2.2 erwähnt, bietet RDF-S3 auch die Möglichkeit der Speicherung von RDF-Daten mit der *Stream Based Parsing*-Methode. Dabei wird kein Speicher-Modell mithilfe von VRP erzeugt, sondern die RDF-Tripel werden direkt gespeichert.

In der bisher getesteten Version von RDF-S3 (v1.7), wurden für jede Datei parallele Threads erzeugt. Dies hatte zur Folge, dass bei Einprozessormaschinen ein Ressourcennutzungsproblem auftrat. Die Threads konkurrierten um die CPU und somit haben sich die Laufzeiten verlängert. In einer modifizierten Version von RDF-S3 (1.8Pre), wurde dieses Problem so umgangen, dass ein neuer Thread erst dann gestartet wurde, wenn der vor ihm gestartete Thread beendet war. Dies garantierte eine sequenzielle Abarbeitung der Dateien und jeder Thread konnte die CPU ungestört von anderen Threads für sich beanspruchen. Bei der ebenfalls getesteten *Stream Based Parsing*-Methode hingegen wird auch die Zeit, die für das Parsen der RDF-Datei benötigt wird, auf die Gesamtzeit angerechnet. Daher

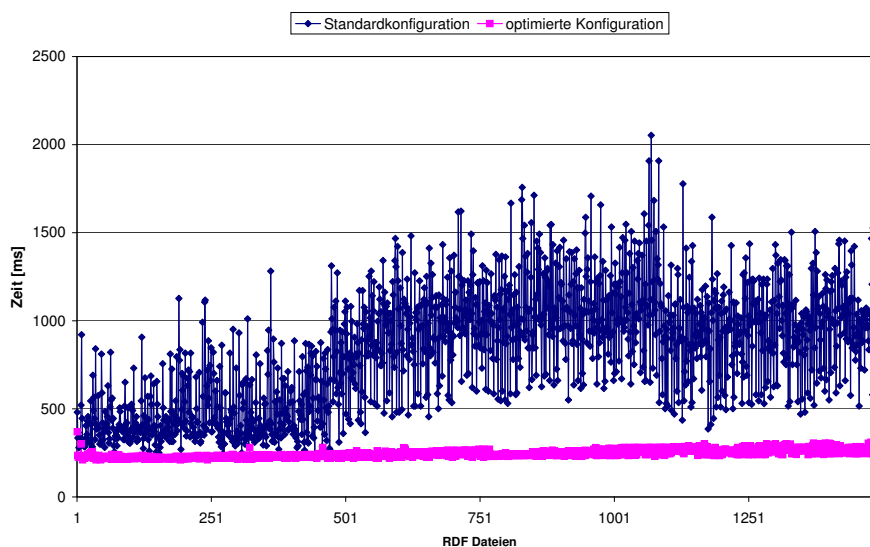


Abbildung 4.21: Standardkonfiguration vs. optimierte Konfiguration (kleine Dateien) – *Stream Based Parsing*-Methode.

sind diese Zeiten nicht mit den Zeiten, die mithilfe von VRP validiert werden, zu vergleichen. Dabei beschränkte sich der Test auf die Gegenüberstellung der Standardkonfiguration und der optimierten Konfiguration.

Auch bei der *Stream Based Parsing*-Methode ist die Performancesteigerung deutlich zu erkennen. Bei der optimierten Konfiguration sind im Vergleich zu der Methode mit dem Speicher-Modell die Zeiten ein wenig höher und nicht vergleichbar konstant, da in ihnen auch die Zeit enthalten ist, die zum Parsen der Datei benötigt und wiederum stärker von den anderen Systemaktivitäten beeinflusst wird. Bei den Tests mit kleinen Dateien betrug der Performancegewinn bei der optimierten Konfiguration gegenüber der Standardkonfiguration im Durchschnitt ca. 63 % (Abb. 4.21). Auch bei den Testreihen mit mittelgroßen bzw. großen Dateien gab es deutliche Performancesteigerungen. Bei der Testreihe mit mittelgroßen Dateien wurde eine Performancesteigerung von ca. 48 % gegenüber der Standardkonfiguration erreicht (Abb. 4.22 auf S.66). Der Performancegewinn bei den Testläufen mit großen Dateien betrug ca. 51 % im Durchschnitt (Abb. 4.23 auf S.66).

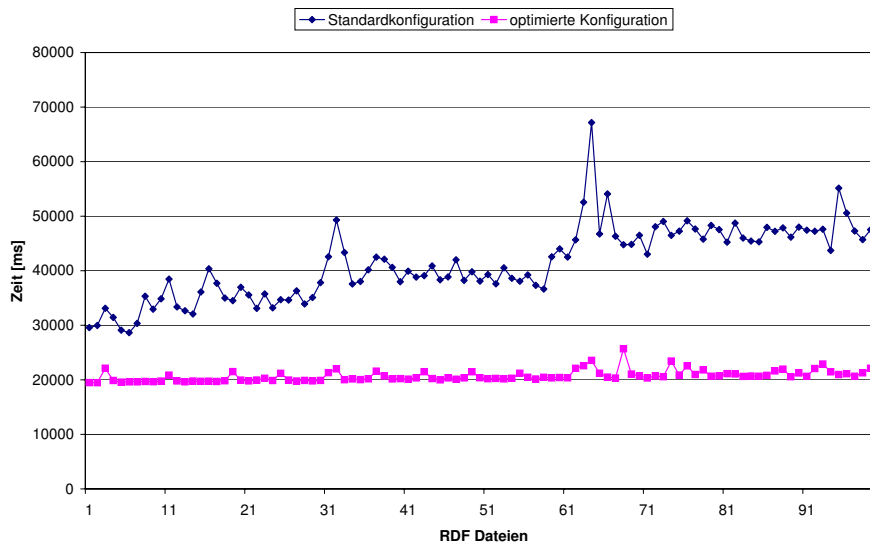


Abbildung 4.22: Standardkonfiguration vs. optimierte Konfiguration (mittelgroße Dateien) – *Stream Based Parsing*-Methode.

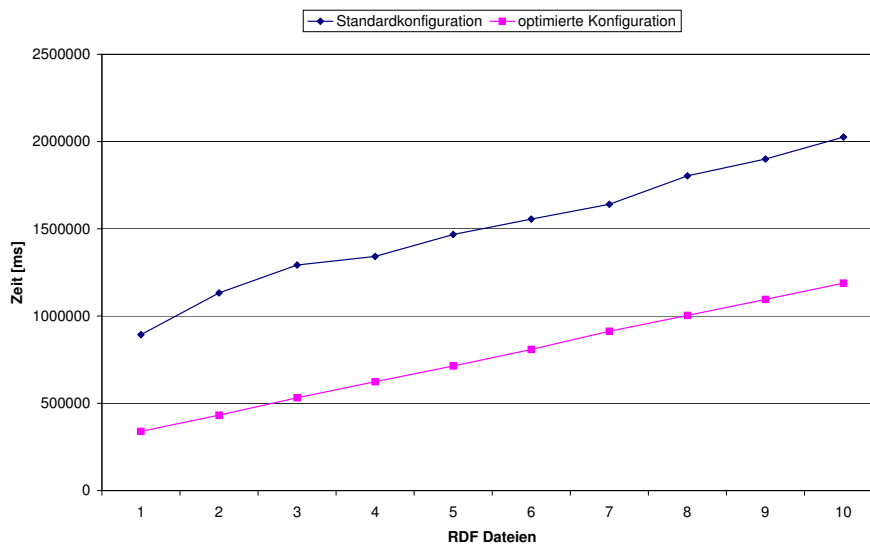


Abbildung 4.23: Standardkonfiguration vs. optimierte Konfiguration (große Dateien) – *Stream Based Parsing*-Methode.

# Kapitel 5

## Zusammenfassung und Ausblick

Im diesem Kapitel wird zunächst die Zusammenfassung der für diese Diplomarbeit durchgeführten Aufgaben und Ergebnisse präsentiert. Anschließend werden einige Empfehlungen für den zukünftigen Einsatz von RDF-S3 gegeben. Mit einem kurzen Ausblick schließt diese Diplomarbeit.

### 5.1 Zusammenfassung

Zu Beginn dieser Diplomarbeit wurde zunächst das semantische Web vorgestellt. Außerdem wurde das Ziel dieser Diplomarbeit festgelegt.

Danach wurde eine Einführung in die Optimierung gegeben. Anschließend wurde die Anwendung RDF-S3 näher betrachtet. Der darauf folgende Abschnitt befasste sich mit dem verwendeten Datenbanksystem - IBM UDB DB2 v8.1. Hier wurden zunächst die Arbeitsweise und Architektur des Datenbanksystems vorgestellt und im Anschluss daran wurden die wichtigsten physischen und logischen Komponenten des Datenbanksystems näher erläutert, die eine entscheidende Rolle bei der Optimierung spielen.

In Kapitel 3 wurden dann zunächst die Vergleichstestmethoden und Werkzeuge beschrieben, die für den Optimierungsprozess eingesetzt werden. Dabei wurden auch kommerzielle Werkzeuge vorgestellt, obwohl sie bei dieser Diplomarbeit selbst nicht zum Einsatz kamen. Der letzte Abschnitt stellte die zugrunde liegende Hard- und Software vor.

Schwerpunkt des Kapitels 4 war die Optimierung der Konfigurationsparameter. Zunächst wurden die Testdaten und die beiden Testmethoden vorgestellt. Dannach wurden die drei Versionen von RDF-S3 miteinander verglichen. In diesem Abschnitt wurde die durch Verbesserungen an der Anwendung RDF-S3 erzielte Performancesteigerung näher betrachtet. Die Performancesteigerung betrug im Durchschnitt ca. 27 %.

Anschließend wurden die Auswirkungen einer Hardwareaufrüstung auf die Performance untersucht. Eine schnellere Festplatte bewirkte dabei einen Performancegewinn von ca. 38 %. In einem großen Abschnitt folgte dann der Bericht über die Konfiguration der wichtigs-

ten Datenbankkonfigurationsparameter. Durch zahlreiche Testläufe wurde eine optimierte Konfiguration gefunden, die gegenüber der Standardkonfiguration eine deutlich bessere Performance zeigte, deren Erreichung das Ziel dieser Diplomarbeit war. Bei den Tests mit kleinen Dateien betrug dieser Performancegewinn ca. 75 % im Durchschnitt, wie im vorherigen Kapitel gezeigt wurde. Der Performancegewinn bei den Tests mit mittelgroßen Dateien erbrachte mit ca. 71 % im Durchschnitt ein ähnliches Ergebnis. Ein vergleichbarer Performancegewinn von ca. 71 % im Durchschnitt wurde auch bei den Tests mit großen Dateien erzielt und somit die Erwartungen übertraf.

Am Ende dieses Kapitels wurde eine Testreihe mit der *Stream Based Parsing*-Methode durchgeführt. Dabei wurde der Vergleich der Standardkonfiguration mit der optimierten Konfiguration näher betrachtet. Dabei betrug der Performancegewinn ca. 63 % im Durchschnitt bei den Tests mit kleinen Dateien. Bei den Tests mit mittelgroßen und großen Dateien hingegen konnte eine Performancesteigerung von ca. 48 % bzw. 51 % erzielt werden.

Die folgende Tabelle fasst die gerade dargestellten Ergebnisse noch einmal zusammen.

Testdaten	<i>VRP-Modell-Methode</i>	<i>Stream Based Parsing-Methode</i>
kleine Dateien	75 %	63 %
mittelgroße Dateien	71 %	48 %
große Dateien	71 %	51 %
Schema-Daten	51 %	55 %

Tabelle. Zusammenfassung der Ergebnisse - Performancegewinn in Prozent<sup>1</sup>.

## 5.2 Empfehlung

Die hier vorgestellte optimierte Konfiguration stellt nur einen Vorschlag für eine Konfiguration im späteren Einsatz dar. Eine derartige Konfiguration der Parameter sollte insbesondere beim Einsatz vergleichbarer Hardware eine solide Grundlage bilden. Für Systeme mit besserer Ausstattung kann sich die im letzten Kapitel vorgestellte optimierte Konfiguration als nicht optimal erweisen. Daher sollte, wenn möglich, vor dem Einsatz der Datenbank auf Systemen mit mehreren Platten insbesondere die Aufteilung der Tabellen auf getrennte unformatierte Laufwerke genügend getestet werden, weil in unserem Falle aus Mangel an Hardware wurde bei der Optimierung der Datenbank auf die Trennung der Tabellen verzichtet. Dies hatte eine leichte Instandhaltung der Datenbank zur Folge, was auch im späteren Einsatz von RDF-S3 nicht unterschätzt werden sollte, denn die Aufteilung der Tabellenbereiche bringt mehr Überwachungsaufwand mit sich.

Bei der Erstellung der Einsatzdatenbank ist weiterhin zu beachten, dass ein 250 MB großer Tabellenbereich für die Benutzerdaten nicht ausreichen wird. Daher sollten die Tabellenbereiche ausreichend groß dimensioniert werden. Es wird aber nicht empfohlen, nur den

<sup>1</sup>Bei der *Stream Based Parsing*-Methode wurde eine modifizierte Version von RDF-S3 (v1.8Pre) verwendet.

Tabellenbereich zu vergrößern, sondern auch zusätzlicher Behälter gleicher Größe zu erzeugen, dies aber nur auf getrennten Platten.

Für einen späteren Einsatz von RDF-S3 ist es wichtig, sowohl die Pufferpoolanzahl als auch die Pufferpoolgröße mit Anzahl und Inhalt der Tabellenbereiche abzustimmen. Insbesondere die Erhöhung der Pufferpoolgröße zeigte eine deutliche Performancesteigerung. Der Wert von 7500 Seiten für die Pufferpoolgröße kann für den späteren Einsatz von RDF-S3 viel zu klein sein. Es sollte daher mit verschiedenen Größen getestet werden.

Wenn das Speichern von grossen RDF-Dateien (RDF-Dateien mit einer sehr großen Anzahl von RDF-Tripel) vorgesehen ist, ist auch dafür zu sorgen, dass die verwendete Hardware die schnelle Abarbeitung der Dateien ermöglicht. Insbesondere wird eine Aufrüstung des Arbeitsspeicher auf 2 GB empfohlen.

## 5.3 Ausblick

Die hier getestete RDF-S3 v1.7 hinterlässt einen sehr guten Eindruck. Insbesondere das Zusammenspiel von RDF-S3 und IBMs DB2 scheint sehr gute Ergebnisse zu liefern. Mit wenig Aufwand ließen sich dabei deutliche Performancesteigerungen erzielen.

Aus Zeitgründen war es bei dieser Diplomarbeit nicht möglich, RDF-S3 mit dem MySQL-Datenbanksystem zu testen. Wenn es zu einem Vergleich von IBMs DB2 und MySQL zu einem späteren Zeitpunkt kommen soll, sollten die hier vorgestellten Standardkonfigurationsergebnisse eine solide Grundlage bilden.

Bei den in dieser Diplomarbeit vorgestellten Datenbankeinstellungen handelt es sich jedoch um Feineinstellungen, die auf die hier verwendete Hardware zugeschnitten sind. Daher kann die Anpassung der Datenbankparameter im Echtlauf – mit viel mehr RDF-Daten und vor allem auf einer anderen Hardware wie z.B. mit mehreren Festplatten – zu einer nicht optimalen Konfiguration führen.

Ein sehr großes Potenzial an Performancesteigerung verbirgt sich selbst in der RDF-S3 Anwendung. Der Performancegewinn von Version zu Version, wie im letzten Kapitel vorgestellt, war auffallend. Die von den Entwicklern von RDF-S3 geplante Parallelisierung der RDF-S3-Arbeitsweise, die der Parallelisierung in DB2 zugute kommen soll, soll in Zukunft eine zusätzliche Performancesteigerung ermöglichen.



# Anhang A

## DB2 Konfigurationsparameter

Die Konfigurationsdateien enthalten Parameter, die Werte definieren, mit denen die Leistung gesteuert werden kann. In DB2 gibt es zwei Arten von Konfigurationsdateien:

- Die Konfigurationsdatei des Datenbankmanagers (*db2system*) für jedes Exemplar und
- Die Konfigurationsdatei für jede einzelne Datenbank (*SQLDBCON*).

In den folgenden Tabellen werden die Konfigurationsparameter aufgelistet, die bei Optimierung sehr wichtig sind. Die Tabellen enthalten die Spalte „Online konfigurierbar“ in der angegeben wird, ob der Parameter online konfigurierbar ist, d.h. ob die Änderungen unverzüglich angewendet werden oder z.B. der Datenbankmanager gestoppt (*db2stop*) und erneut gestartet (*db2start*) werden muss, damit die Änderungen wirksam werden. In der Spalte „Leistungsrelevanz“ ist vermerkt, in welchem Ausmaß sich jeder Parameter in Bezug auf die Systemleistung auswirken kann:

- **Hoch.**– Der Parameter kann wesentliche Auswirkungen auf die Leistung haben.
- **Mittel.**– Der Parameter kann unter Umständen Auswirkungen auf die Leistung haben.
- **Niedrig.**– Der Parameter hat weniger bedeutende Auswirkungen auf die Leistung.
- **Keine.**– Der Parameter hat keine direkte Auswirkungen auf die Leistung.

In den folgenden Tabellen sind alle aufgelisteten Parameter, nach [DB2-02], mit „Hoch“ eingestuft. Gleichzeitig alle diese Parameter unterstützen nicht das Schlüsselwort *automatic*, d.h. DB2 kann nicht die Parameterwerte automatisch, je nach Ressourcenanforderungen, anpassen. Aus diesen Gründen beide Spalten „Automatik“ und „Leistungsrelevanz“ werden in den Tabellen nicht angezeigt<sup>1</sup>.

---

<sup>1</sup>Für alle hier aufgelisteten Parameter gilt: *automatic=Nein* und *Leistungsrelevanz=Hoch*.

Parameter	Online Konfigurierbar	Zusatzinformationen
<i>agentpri</i>	Nein	Agentenpriorität
<i>aslheapsz</i>	Nein	Zwischenspeichergröße für Anwendungsunterstützungsebene
<i>audit_buf_sz</i>	Nein	Prüfpuffergröße
<i>intra_parallel</i>	Nein	Partitionsinterne Parallelität aktivieren
<i>java_heap_sz</i>	Nein	Maximale Zwischenspeichergröße für Java-Interpreter
<i>maxagents*</i>	Nein	Maximale Anzahl von Agenten
<i>maxcagents*</i>	Nein	Maximale Anzahl gleichzeitig aktiver Agenten
<i>max_querydegree</i>	Ja	Maximaler Grad der Parallelität bei Abfragen
<i>num_poolagents</i>	Nein	Agentenpoolgröße
<i>rqrioblk</i>	Nein	E/A-Blockgröße für Clients
<i>sheapthres</i>	Nein	Schwellenwert für Sortierspeicher

Tabelle 1. Konfigurierbare Konfigurationsparameter des Datenbankmanagers.

Parameter	Online Konfigurierbar	Zusatzinformationen
<i>avg_appls</i>	Ja	Durchschnittliche Anzahl aktiver Anwendungen
<i>catalogcache_sz</i>	Ja	Katalogcachegröße
<i>chnpggs_thresh</i>	Nein	Schwellenwert für geänderte Seiten
<i>dft_degree</i>	Ja	Grad der Parallelität
<i>locklist**</i>	Ja	Maximaler Speicher für Sperrenliste
<i>logbufsz</i>	Nein	Protokollpuffergröße
<i>maxappls*</i>	Ja	Maximale Anzahl aktiver Anwendungen
<i>maxlocks**</i>	Ja	Maximale Anzahl Sperren/Anwendung
<i>mincommit</i>	Ja	Anzahl der Gruppenfestschreibungen
<i>num_iocleaners</i>	Nein	Anzahl asynchroner Seitenlöschfunktionen
<i>num_ioservers</i>	Nein	Anzahl von E/A-Servern
<i>pckcachesz</i>	Ja	Größe des Paketcache
<i>seqdetect</i>	Ja	Markierung für Sequenzerkennung
<i>sheapthres_shr</i>	Nein	Schwellenwert für Sortierspeicher für gemeinsame Sortiervorgänge
<i>sortheap</i>	Ja	Zwischenspeichergröße für Sortierlisten

Tabelle 2. Konfigurierbare Konfigurationsparameter für die Datenbank.

**Anmerkung:** Die mit \* markierten Parameter sind in Bezug auf die Leistungsrelevanz mit „Mittel“ eingestuft [DB2-02] und die mit \*\* markierten Parameter haben hohen Einfluss auf die Eskalation.

# Anhang B

## Inhaltsbeschreibung der CD

Die Diplomarbeit liegt in zwei verschiedenen Formaten vor, sowohl als *Diplomarbeit.pdf* als auch als *Diplomarbeit.ps*

Das Verzeichnis *Testdaten* enthält die Testdateien:

- *culture.rdf*
- *culture-data.rdf*
- *chefmoz.guides-mod.rdf* und
- *PerfTest.rdf*

Das Verzeichnis *RDF-S3* enthält die verschiedenen Versionen der Anwendung RDF-S3.

Das Verzeichnis *Ergebnisse* enthält die Ergebnisse für die Standardkonfiguration und optimierte Konfiguration.



# Literaturverzeichnis

- [RDF-S3] RDF-S3 Homepage, online:  
<http://www.dbis.cs.uni-frankfurt.de/~tolle/RDF/RDFS3/>
- [RDF] W3 Homepage, online:  
<http://www.w3.org/TR/rdf-primer/> (August 2004)
- [DB2-00] (2000) IBM DB2 Universal Database v7.1 *Performance Tuning Guide*. IBM Red-Book.
- [DB2-02] IBM DB2 Universal Database. Systemverwaltung: Optimierung *Version 8*. Teil der Dokumentation von DB2.
- [DB2-02a] IBM DB2 Universal Database. System Monitor Guide and Reference *Version 8*.
- [DB2-D] Dey, J. (2004) *DB2 UDB v8.1 Performance Tuning Guide*.
- [DB2-S] Steinbach, T. *DB2 Performance & Tuning. A Brief Overview*.
- [DB2-Z] Zikopoulos, P.C. *DB2 Planning*. Online: <http://www7b.software.ibm.com/dmdd/> (Juni 2004)
- [JB] Behl, J. (2003). *Performance-Optimierung einer Datenbankanwendung auf IBMs DB2 UDB*.
- [S02] Skoumal, D. (2002) *Generierung von RDF-Daten aus Webbasierten Informationsquellen*.
- [T00] Tolle, K. (2000). Analyzing and Parsing RDF. Master's Thesis, University of Hannover.
- [T04] Tolle, K. (2004). AISTA'04 *Understanding data by their context using RDF*. Online: <http://www.dbis.cs.uni-frankfurt.de/~tolle/Publications/2004/>
- [TW04] Tolle, K. and Wleklinski, F. (2004). *Trust and context using the RDF-Source related Storage System (RDF-S3) and easy RQL (eRQL)*.
- [TW04a] Tolle, K. and Wleklinski, F. (2004). *RDF-S3 und eRQL: RDF Technologien für Informationsportale*.

- [YL01] Yevich, R. und Lawson, J. (2001). *DB2 High Performance Design and Tuning*. Prentice Hall.
- [Z03] Zabotoczky, J. (2003). *Speicherung von RDF-Daten*.
- [JM] Java Magazin. Ausgabe 6, 2003.
- [JU] JUnit Homepage, online: <http://www.junit.org> (September 2004)
- [JUP] JUnitPerf Homepage, online: <http://www.clarkware.com/software/JUnitPerf.html> (September 2004)
- [MI] LoadRunner Homepage, online: <http://www.mercuryinteraktive.de/products/loadrunner> (September 2004)
- [iX 3/05] Moch, F. ‘‘Vergleich der Speicherformen von DB2-Datenbanken – Besser nicht roh’’. iX - Magazin für Professionelle Informationstechnik. Ausgabe 3/2005.