Problemseminar Schema–Evolution: Schema–Evolution in kommerziellen DBS (dynamische Schema-Evolution)

Jáno Gebelein

5. Dezember 2005

Vorwort

Diese Arbeit entstand an der Universität Leipzig – Institut für Informatik – Fakultät für Datenbanken im Zusammenhang mit der Durchführung eines Problemseminares über die Schema–Evolution in Datenbanksystemen. Jeder Teilnehmer hatte sich dabei auf eine frei wählbare Problematik aus dem Themengebiet festzulegen und diese in Verbindung mit einer 60 minütigen Präsentation vorzustellen.

Inhaltsverzeichnis

Kapitel 1

Einleitung und Motivation

[WIK05] definiert Schema als „formales Modell der Struktur von Daten mit der Aufgabe der Definition von Relationen als Tupel von Attributen, denen in vielen Fällen Datentypen zugewiesen sind“. Dabei reicht ihre Komplexität von einfachen Attributlisten bis hin zu komplexen Ontologien. Bei Datenbank-Schemas handelt es sich also um die Gruppierung von Tabellen, in denen Daten abgelegt werden können. Sind diese Schemata einmal spezifiziert und in die Praxis umgesetzt oder gar schon über einen längeren Zeitraum erfolgreich im Einsatz, so führt der Wunsch nach Änderung bzw. Anpassung oft ins Leere. Als Grund hierfür wird im Allgemeinen der zu niedrige Kosten-Nutzen-Faktor genannt.
Viele Anbieter kommerzieller Datenbanksysteme haben diese Anforderung des Marktes erkannt und versucht, sie in ihr bestehendes System zu integrieren, hierzu gehören unter Anderem [IBM04] IBM DB2 (Kapitel 3), [ODB05b] Oracle Database (Kapitel 4) und [MSS05a] Microsoft SQL-Server (Kapitel 5). Unter der wissenschaftlichen Bezeichnung „Schema Evolution“ geführt, beschäftigt sich dieses Gebiet mit den Voraussetzungen, Auswirkungen und der Durchführbarkeit von Änderungen bestehender Datenbank-Schemas zunehmender Komplexität und den Versuchen, dem Kunden ein bedienbares Werkzeug in die Hand zu legen, mit dem er selbstständig sicher und effizient in dieser Hinsicht tätig werden kann.
Diese Arbeit soll nun die gerade genannten Punkte in Bezug auf ihre Umsetzung durch die Hersteller der großen Datenbanken näher beleuchten und einen Einblick in den Stand der Technik, welcher aktuell auf dem Software-Markt verfügbar ist, geben.

Kapitel 2

Prior Art – DROP'n'reCREATE

In der heutigen Zeit werden beinahe täglich neue Datenbanken mit zugehörigen Schemas erstellt. Damit verbunden ist die Schaffung entsprechender Anwendungssoftware die darauf aufsetzt und der die in den Tabellen enthaltenen Daten zugrunde liegen. Das fertiggestellte System zeigt dann mehr oder weniger die geforderten Eigenschaften und sein Lebenszyklus in der freien Marktwirtschaft beginnt. Nicht selten werden die Anforderungen in der folgenden Zeit überarbeitet und eventuell veränderten Bedingungen angepasst. Hier beginnt dann die Arbeit des informationstechnischen Personals.
Jeder erfolgreiche Datenbank-Administrator sah sich schon mindestens einmal in seinem Leben der Situation einer notwendigen Änderung eines bestehenden Datenbank-Schemas gegenüber und jeder dieser Personen kennt die Problematik, der damit verbundenen Abläufe. Viele der erwünschten Anpassungen konnten bisher aber eben nicht einfach „mal eben" durch ein ALTER gelöst werden, sondern bedürfen umfangreicher DROP und CREATE Anweisungen an das DBMS und führten im Allgemeinen immer wieder zu den gleichen Problemen: nämlich einerseits der dafür meist notwendigen, in größeren Unternehmen recht kostspieligen, down-time der Datenbank und andererseits den nun wieder in erhöhtem Maße möglichen Fehlerquellen und der damit einhergehenden Testphase.
Allein bei Änderung einer Tabellenspalte von CHAR(10) auf CHAR(15) nennt [MUL04] die folgenden, notwendigen Schritte:

1. Unload the data, extract the DDL and authorizations for the table you are about to change and all dependent objects, (indexes, views, synonyms, triggers, and so on)

2. Drop the table

3. Modify the DDL for the table change the length of the particular column from 10 to 15

4. Run the CREATE statement to re-create the table

5. Run the CREATE statements for the dependent objects (indexes, views, synonyms, triggers, etc.)

6. Re-build the authorizations for the table by running the appropriate GRANT statements
7. Re-create all dependent objects and re-build their authorizations
8. Reload the data taking care to build the LOAD statement properly because there is a new column right smack dab in the middle of the other columns
9. Don't forget to run RUNSTATS to gather statistics for optimization and run COPY to backup the data in its new format after it is loaded
10. REBIND all affected plans and packages
11. Test everything to make sure your changes were implemented correctly

An diesem Beispiel ist deutlich zu erkennen, wie zeit- und damit auch kostenaufwendig sich eine solch geringfügige Änderung gestalten kann. Hinzu kommt noch die Nichtverfügbarkeit der Daten während dieses gesamten Zeitraums, was die Kosten noch weiter in die Höhe treibt. Scheitert auch nur ein einziger Zwischenschritt oder wurde er komplett vergessen, so sind weitere Schwierigkeiten vorprogrammiert.
Um bisher Anpassungen dieser Dimension besser in den Griff zu bekommen, bieten verschiedene Drittanbieter für unterschiedliche Datenbanksysteme Werkzeuge, die unter Einsatz von menüunterstützten Strukturen das Auffinden und Modifizieren aller notwendigen Änderungen innerhalb der Datenbank erleichtern jedoch nicht vereinfachen. Denn diese setzen auch nur auf die manuell mögliche DROP'n'reCREATE Technik auf.

Kapitel 3

IBM DB2 Universal Database

3.1 prior to version 8 - Change Management

Bisherige Versionen der IBM Datenbank boten eine schier endlose Menge an Funktionalitäten, die den Administratoren mehr oder weniger wertvolle Dienste leisteten. Genau diese Tools ließen einen jedoch manchmal im Regen stehen, wenn es schlichtweg darum ging, ein bereits bestehendes Schema, wenn möglich zur Laufzeit, nach den eigenen Vorstellungen und Wünschen zu modifizieren.
Die Antwort von IBM ist nach einer zunächst längeren Entwicklungsphase das neue Feature mit dem Namen „Online Schema Evolution“, das auf das bisher existierende „Change Management“ aufsetzt. Was bisher unumgänglich über DROP'n'reCREATE (siehe 2) mehr oder weniger anwenderfreundlich gelöst werden musste, kann nun effizient und zur Laufzeit durch einfaches Ändern der entsprechenden Einträge mit Hilfe der neu gebotenen Basisfunktionen erfolgen. „What customers would like is the ability to alter“ [DBA05]. So ist es zum Beispiel mit Hilfe des bereits bekannten ALTER Statements möglich, die Länge von Zeichenketten dynamisch zu verändern, wobei alle dazu notwendigen Anpassungen der Speicher- und Verwaltungsstrukturen vom Datenbanksystem automatisch durchgeführt werden.
Wenige der bereits mit Version 7 möglichen DDL ALTER Operationen, wie zum Beispiel das Umbenennen einer Tabelle, Hinzufügen einer Spalte am Tabellenende oder die mögliche Abänderung der Länge eines VARCHAR nach oben, ohne dazu die Datenbank anhalten zu müssen, hatten zwar bereits den Grundgedanken der heutigen Schema Evolution, mussten jedoch teuer mit down-time bezahlt werden.
Abbildung 3.1 zeigt deutlich den verbesserten Ablauf und die Verfügbarkeit der Daten bei Anpassung des dazugehörigen Schemas, durch den sich in Version 8 nun wertvolle Zeit einsparen lässt.
Zusammenfassend waren also, um nur die Wichtigsten zu nennen, durch das bisherige Change Management bereits die folgenden Anpassungen innerhalb der laufenden Datenbank möglich:

- Hinzufügen einer Spalte am Tabellenende ohne dazu die gesamte Tabelle oder Programme, die auf sie zugegriffen haben, sperren zu müssen

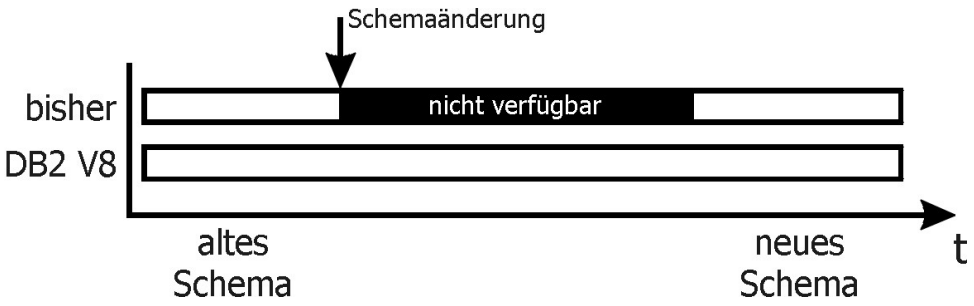

Abbildung 3.1: Datenverfügbarkeit bei Schemaänderung V7 vs. V8

- Umbenennen einer Tabelle, ohne sie dazu per DROP'n'reCREATE (siehe 2) neu anlegen zu müssen
- das Verwenden des ALTER Statements, um die Länge einer VARCHAR Spalte nach oben (nicht nach unten) zu korrigieren
- Partitionsgrenzen zu ändern
- Anwendungsänderungen über Paketversionierung zu managen
- in einem „DB2 data share complex" Änderungen am Code durch PTFs durchzuführen, während andere Mitglieder aktiv waren
- das Verwenden der REORG und LOAD RESUME Utilities zur Laufzeit, während andere Workloads mit den neu zu organisierenden oder zu ladenden Daten arbeiteten

Dies ist jedoch nur ein kleiner Ausschnitt aus dem tatsächlichen Funktionsumfang der DB2 UDB, von dem einige Änderungen natürlich nicht so einfach durchzuführen waren.

3.2 Wesentliche Neuerungen in V8

Mit jeder neuen Version ihrer Datenbank bietet IBM seinen Kunden eine größere und vielfältigere Programmfunktionalität auf Basis des stabilsten und zuverlässigsten Systems, das es auf dem Markt gibt: ihren aktuellen Mainframe Systemen mit z/OS 1.3+. Auch in der aktuellsten Version 8 wurden gegenüber der Vorgängerversion 7 verschiedene Änderungen bzw. Erweiterungen vorgenommen. Vor allem das viel umworbene neue Feature der „Online Schema Evolution", „Online Schema Change" oder „Online Alter", wie es bei IBM genannt wird, bringt einige Verbesserungen im Bereich der Schema-Anpassung zur Laufzeit mit sich ([IBM04]). Unter Anderem sind dies:

- Vergrößern der Länge einer CHAR(n) Spalte
- Typänderung innerhalb von Character Datentypen (CHAR, VARCHAR)

- Typänderung innerhalb von numerischen Datentypen (SMALLINT, INTEGER, FLOAT, REAL, FLOAT, DOUBLE, DECIMAL)
- Typänderung innerhalb von graphischen Datentypen (GRAPHIC, VARGRAPHIC)
- Datentypänderungen in durch Views referenzierten Spalten
- diese Änderungen sind auch für Spalten möglich, die Teil eines Index sind
- Hinzufügen von Spalten zu einem Index
- den Partitionierungsindex löschen oder eine neue Tabelle ohne selbigen anlegen
- den Clusterindex ändern
- erzeugen/ändern eines Index, sodass dieser über nicht ausgefüllte char-Spalten variabler Länge innerhalb eines Schlüssels verfügt
- ändern von „identity columns“
- hinzufügen einer Partition am Tabellenende, die den Grenzwert vergrößert
- Rotation von Partitionen
- automatisches Rebalancing von Partitionen durch REORG
- REORG von Teilen, die schon für ein REORG anstehenden
- lockern der Restriktionen von Indexen, die sich gerade in Zuständen eines anstehenden Recover oder Rebuild befinden

Im Folgenden (siehe 3.3) soll nun ein Einblick in einige ausgewählte Features gegeben werden.

3.3 DB2 UDB im Detail

3.3.1 Änderung von Datentypen

Bedingungen

Manchmal ist es notwendig, den Datentyp einer Spalte innerhalb einer bereits existierenden Tabelle nachträglich zu ändern. In Versionen kleiner als 8 konnte dies nur durch ein löschen und anschließendes wiedereinspielen der Tabelle (siehe 2) gelöst werden. Die neue Version 8 erlaubt es nun, die bereits vorliegenden Daten einer zu ändernden Tabellenspalte zur Laufzeit zu ändern, ohne dabei die Gültigkeit der bereits enthaltenen Daten zu verlieren. Dies setzt voraus, dass es sich um einen charakter oder numerischen Datentyp handelt und dass der neue Datentyp größer ist, als der bereits vorhandene, da sonst wertvolle Informationen abgeschnitten würden.

Syntax

aktueller Datentyp	möglicher neuer Datentyp
SMALLINT	INTEGER, REAL, DOUBLE, >=DECIMAL(5,0)
INTEGER	DOUBLE, >=DECIMAL(5,0)
REAL (oder FLOAT(4))	DOUBLE (oder FLOAT(8))
<=DECIMAL(15,m)	DOUBLE
DECIMAL(n,m)	DECIMAL(n+x,m+y)
CHAR(n)	CHAR(n+x), ARCHAR(n+x)
VARCHAR(n)	CHAR(n+x), VARCHAR(n+x)
GRAPHIC(n)	GRAPHIC(n+x), VARGRAPHIC(n+x)
VARGRAPHIC(n)	GRAPHIC(n+x), VARGRAPHIC(n+x)

Tabelle 3.1: mögliche Datentypkonvertierung

[MUL04] zeigt in Tabelle 3.1 übersichtlich die Kompatibilität der einzelnen Datentypen zueinander. Eine solche Änderung kann einfach über das bereits bekannte ALTER TABLE Statement vorgenommen werden, wobei folgender Syntax zu verwenden ist:

```
ALTER TABLE Tabellenname
            ALTER [COLUMN] Spaltenname
            SET DATATYPE Datentyp
```

Auswirkungen

Nachdem dem DBMS die Änderung mitgeteilt wurde, erstellt dieses eine neue Version der Tabelle (siehe 3.3.4). Es verbleiben zunächst alle bereits eingetragenen Daten in ihrer bisherigen Version, für neue Einträge gilt jedoch bereits der gewünschte Datentyp. Wird nun ein Datensatz mit, noch im alten Format vorliegenden Einträgen angefordert, so werden die entsprechenden Daten in das neue Format konvertiert und zurückgegeben. Wird eine bestehende Zeile jedoch geändert oder neu eingefügt, so werden die entsprechenden Daten in das neue Format konvertiert und in die Datenbank übernommen. Erst nach einer Reorganisation der kompletten Tabelle werden alle Einträge in das neue Format übernommen und die Umwandlung des Datentyps ist vollständig. Die noch vorhandene alte Version der Daten bleibt jedoch noch so lange erhalten, bis keinerlei Verweise von noch nicht aktualisierten Objekten mehr bekannt sind.

Einschränkungen

Dies ist auch der Grund, weshalb ein Datentyp nur vergrößert werden kann, jedoch nicht verkleinert und weshalb es sich um einen charakter bzw. numerischen Datentyp handeln muss. Es ist (noch) nicht möglich, Datentypen wie ROWID, DATE, TIME, TIMESTAMP, oder LOB zu verändern. Weiterhin können keine Anpassungen der Spalte unter den folgenden Umständen durchgeführt werden:

- als Teil einer „materialized query table“
- als Teil eines „referential constraint“
- bei Definition als IDENTITY
- bei Definition einer FIELDPROC

Die Änderung des Schemas zieht jedoch noch weitere Kreise. Speziell auf die Performance der Datenbank bezogen, führt sie zu einer erweiterten Versionsverwaltung und dadurch nötigen, umfangreicheren Protokollierungsmaßnahmen, was die Timings verschlechtert. Deshalb wird empfohlen, nach einer erfolgreichen Schema Evolution stets eine Reorganisation der Tabelle durchzuführen.
Vorsicht ist weiterhin geboten bei von den Daten abhängigen Objekten: Plans, Packages und Cached Dynamic Statements, die die geänderte Tabelle referenzieren, verlieren ihre Gültigkeit und werden nur umgeleitet, sofern für sie auto-rebind aktiviert wurde.
Anwendungsprogramme, die geänderte Spalten referenzieren, sei es durch statisches oder dynamisches SQL, werden unter Umständen ungültig. Besonders bei statischem SQL, bei dem definierte Variablen fest in Verbindung mit einer Spalte stehen, muss eine manuelle Anpassung vorgenommen werden, um den vergrößerten Datentyp aufnehmen zu können, der andernfalls einfach abgeschnitten wird.
Auch Views, die geänderte Spalten enthalten müssen geupdated werden, dies geschieht jedoch automatisch mit dem Absetzen des ALTER TABLE Statements. Scheitert einer dieser Updates, so scheitert die gesamte Änderungsoperation und die Datenbank wird zurückgesetzt. Auch hier werden nun alle Plans, Packages und Cached Dynamic Statements auf den Views ungültig. Da Views selbst wiederum Views enthalten können, läuft hier unter Umständen ein rekursiver Prozess an.
Für check Bedingungen gilt dasselbe: sie werden automatisch geändert und eventuell dadurch entstandene Fehler führen zum Abbruch der gesamten Operation.
Ein wesentlicher Punkt muss auch bei der Verwendung von Indexen beachtet werden. Beinhalten diese nämlich Spalten, die gealtert wurden, so werden sie zwar automatisch angepasst, ihre zukünftige Verfügbarkeit hängt jedoch von der Art der durchgeführten Änderung ab. Sofort verfügbar sind Indexe, wenn CHAR, VARCHAR oder VARGRAPHIC in ihrer Länge vergrößert werden, Verzögerungen entstehen bei numerischen Längenanpassungen, wie sie bei SMALLINT, INTEGER, DECIMAL, NUMERIC, FLOAT, REAL, und DOUBLE entstehen. Gerade bei Indexobjekten können hier die Performancelevel einbrechen. Es sollte also im Anschluss stets eine Reorganisation gestartet werden.

3.3.2 Änderung von Indexen

Um schneller auf die Daten zugreifen zu können, wird häufig ein Tabellenindex verwendet. Wenn es jedoch notwendig wird, diesen zu ändern, so war dies bisher auf wenige seiner Eigenschaften beschränkt. Seit Version 8 ist es nun auch möglich, zusätzliche Indexattribute zur Laufzeit hinzuzufügen oder bereits bestehende Einstellungen abzuändern, ohne

den Index dabei vom Netz nehmen zu müssen. Auch dafür kann das bereits bekannte ALTER Statement verwendet werden.

Syntax

Um einem Index eine neue Spalte am Ende des bereits bestehenden Key hinzuzufügen, ist folgender Syntax zu verwenden:

```
ALTER INDEX Indexname
            ADD COLUMN (Spaltenname [ASC | DESC])
```

Um einen bestehenden Index zu verändern, ist folgender Syntax zu verwenden:

```
ALTER INDEX Indexname
            [(NOT) CLUSTER | (NOT) PADDED]
```

Dabei bietet CLUSTER die Möglichkeit der Anpassung von Clusterinformation und PADDED die Festlegung auf die Ausweitung der Indexeinträge.

Bedingungen, Auswirkungen und Einschränkungen

Neue Spalten können nun bereits bestehenden Indexe hinzugefügt werden, ein Update auf einem nicht existenten Index führt also nicht zu dessen Erstellung, sondern bricht ab. Weiterhin kann die Reihenfolge der Keys durch die Verwendung von ALTER INDEX nicht verändert werden, im speziellen ist es nicht möglich, einen hinzugefügten Key an den Anfang des Index zu stellen. Bei Betrachtung der Performance ist noch zu erwähnen, dass Indexerweiterungen, bei denen die neue Spalte vor dem COMMIT gleichzeitig zu Tabelle und Index hinzugefügt wird, zu keinen Performanceverlusten führen, während ein Index, dem eine bereits in einem Tabellenschema eingetragene Spalte hinzugefügt wird, für optimale Leistung erst einer Reorganisation bedarf.
Bei Änderung der Clusterinformationen muss darauf geachtet werden, dass jeweils nur ein Clusterindex erlaubt ist. Soll dieser Index also wechseln, so muss er beim Alten entfernt und beim Neuen hinzugefügt werden, jeweils durch ein eigenes COMMIT bestätigt. Ist der Wechsel erfolgreich, so werden INSERT Anweisungen nach der neuen Reihenfolge geclustert; bereits bestehende Einträge werden nicht verändert, bevor nicht eine separate Reorganisation durchgeführt wurde.
Das Verwenden von PADDING weist das DBMS an, eine Indexspalte eines Datentyps variabler Länge auf seine maximal mögliche Ausdehnung zu vergrößern. Dies war in den Programmversionen vor V8 der Stand der Dinge, kann nun jedoch auch durch Voranstellen eines NOT verhindert werden. Dies führt dazu, dass nur die tatsächliche, variable Länge im Index gespeichert wird.

3.3.3 Operationen mit Partitionen

Seit IBM UDB V8 ist es nun auch möglich, Partitionen innerhalb von Tablespaces dynamisch anzulegen, zu rotieren, zu rebalancieren und ihre Grenzen zu setzen. Dies gestaltete sich bisher eher von problematisch bis hin zu unmöglich und konnte nur durch

DROP'n'reCREATE (siehe 2) des Tablespace erfolgen. Dabei mussten in besonderem Maße sämtliche bereits vorliegende Daten zuvor gesichert und anschließend wieder eingefügt werden, was bei größeren Beständen nahezu unmöglich war.

Bedingungen

Um in den Genuss dieser neuen Features zu kommen, muss lediglich von indexorientiert auf tabellenorientiert partitionierte Tabellen gewechselt und der Tablespace angehalten werden. Danach ist auch hier die Verwendung des bekannten ALTER Statements möglich, um Modifikationen an den Partitionierungseinstellungen vorzunehmen.

Syntax

Um einem bestehenden Tablespace mit einer oder mehreren Partitionen eine neu angelegte Partition hinzuzufügen, ist folgender Syntax zu verwenden:

```
ALTER TABLE Tabellenname
     ADD PARTITION ENDING AT (constant)+
```

Hierbei bestimmt das ENDING AT (constant,...) die obere Grenze der Schlüsseleinträge der neuen Partition, die natürlich größer denen aller anderen Partitionen innerhalb der Tabelle sein sollte.
Für die Rotation gilt:

```
ALTER TABLE Tabellenname
         ALTER PART ROTATE (FIRST TO LAST)
         VALUES (constant)+ RESET (REUSE)
```

RESET führt dabei zum Löschen aller in der Partition enthaltenen Daten.
Zur Anpassung von Partitionsgrenzen wird folgender Syntax verwendet:

```
ALTER TABLE Tabellenname
         ALTER PART integer VALUES (constant)+
```

Der Integer-Wert gibt hier die Nummer der zu ändernden Partition an.

Auswirkungen

Nachdem eine neue Partition hinzugefügt wurde, sollte ALTER TABLESPACE und ALTER INDEX ausgeführt werden, um alle Abhängigkeiten zu korrigieren. Zu erwähnen ist weiterhin, dass alle der Tabelle zugehörigen Packages, Plans und Dynamic Cache Statements ihre Gültigkeit verlieren. Besitzt die letzte, zuvor angelegte, Partition noch keinen oberen Rahmen, so wird dieser mit dem Anlegen der neuen erzwungen und beide Partitionen müssen reorganisiert werden.
Beim Rotieren einer Partition wird selbige komplett geleert (Vorsicht!) und steht fortan am Ende der letzten als neue Partition für die Neuaufnahme von Daten zur Verfügung.

Einschränkungen

Neue Partitionen können fortlaufend angelegt werden, bis das für den Tablespace zulässige Maximum erreicht ist, dabei ist nur zu beachten, dass pro COMMIT jeweils nur maximal eine neue Partition hinzugefügt werden kann. Ist es also beabsichtigt gleichzeitig mehrere Partitionen hinzuzufügen, so muss zwischen jeder ein separates COMMIT abgesetzt werden. Dies gilt auch für das Rotieren. Zusätzlich kann jeweils nur die erste, also die älteste, Partition rotiert werden und an das Ende aller Anderen gebracht werden, daher findet sich auch im zugehörigen Syntax keine Abfrage der Partitionsnummer.

3.3.4 Versionierung in DB2

Alle durch Schema Evolution nun möglichen Anpassungen machten es notwendig, Objekte in mehreren Zuständen erfassen zu können. Daher wurde in die Architektur der Datenbank das Konzept der Objekt-Versionierung integriert. Zwar gab es dieses bereits in Vorgängerversionen, jedoch nur bezogen auf Indexe mit VARCHARs. Da sich nun komplette Datentypen ändern können, diese aus Performancegründen jedoch nicht komplett live geupdated werden sollen, ist es unumgänglich, auf die entsprechenden Daten sowohl im alten, wie auch im neuen Format zugreifen zu können. Auch ist es nicht möglich, sämtliche bereits vorliegenden Abhängigkeiten innerhalb anderer Objekte gleichzeitig auf das neue Format umzustellen und so müssen diese vorübergehend mit den alten Einstellungen Vorlieb nehmen. Es kann so trotz unterschiedlicher Formate von Tabellen, Tablespaces und Indexe maximale Verfügbarkeit erreicht werden. Versionsgrenzen wurden für Indexe bei 16 und Tablespaces bei 256 gesetzt. Dabei zählen jedoch nur aktive Versionen, die von anderen Objekten auch tatsächlich verwendet werden.

Versionsgenerierung

Eine neue Version wird in DB2 in folgenden Situationen vergeben:

- Änderung des Datentyps einer Spalte
- Hinzufügen bei Indexe
- Ändern von Indexen

Dabei werden mehrere, durch lediglich ein COMMIT abgeschlossene, Datentypänderungen in einer einzigen Version gespeichert.

3.4 To Be Continued ...

Ob mehrere hundert Attribute, Partitionsnummern oder Datentypen zum Wechsel anstehen spielt ab Version 8 nun keine Rolle mehr, alles bei voller Performance, Skalierbarkeit und Verfügbarkeit. Änderungen können viel flexibler in weniger Schritten und dadurch mit weniger down-time realisiert werden. IBM hat sich mit Online Schema Evolution

das Ziel gesetzt, in einigen Jahren sämtliche durchzuführende Änderungen während der Laufzeit und bei voller Datenverfügbarkeit durchführen zu können. In V8 wurde der erste Schritt in diese Richtung gegangen und IBM plant für die Zukunft, dieses Feature in den kommenden Programmversionen stetig zu verbessern und zu erweitern, um so noch stärker auf die Anforderungen seiner Kunden einzugehen.

Kapitel 4

Oracle Database

Auch Oracle hat die Anforderungen des Marktes verstanden und mit Veröffentlichung seiner Datenbank in Version 9i damit begonnen, Anpassung nicht mehr nur hinsichtlich der Datenreorganisation, sondern auch hinsichtlich der Datenredefinition zuzulassen. Beginnend mit online Funktionalitäten wie Indexerzeugung, Indexrebuilding, Indexzusammenführung und der Möglichkeit, indexorientierte Tabellen (IOT) online zu verschieben, wurden in vorherigen Versionen die dafür notwendigen Grundsteine gelegt. Durch das neue Feature der Datenreorganisation ist nun auch die physische Manipulation von Daten- und Tabellenstrukturen gegeben, ohne dabei die gesamte Datenbank oder auch nur Teile davon zeitweise unerreichbar werden zu lassen.

4.1 Wesentliche Neuerungen

Zu den wesentlichen Neuerungen innerhalb der Oracle Database, die alle eine erhöhte Verfügbarkeit, Wartbarkeit, Performance, Speicherplatzausnutzung und akzeptable Antwortzeiten zum Ziel haben, gehören unter Anderem:

- Konvertieren von LONG und LONG RAW Spalten in ein LOB
- Nutzung von eindeutigen Schlüsseln als Alternative zu Primärschlüsseln oder RowID
- die Möglichkeit, Spalten zu sortieren
- Tabellentausch ohne Recompilierung von Stored Procedures
- Einschrumpfen von Segmenten
- Reorganisation einzelner Partitionen
- Reorganisation erweiterter Queue und geclusterter Tabellen
- Reorganisation von table constraints

- Kopieren abhängiger Objekte, wie Trigger, Constraints und Indexe
- Erweiterte Funktionalitäten hinsichtlich Kopieren abhängiger Objekte, wie das Übernehmen oder Kopieren von Statistiken, die Überprüfung und Nicht-Nullierung von Constraints oder der Support für Objektabhängigkeiten von nested tables

All diese Features können unter Verwendung der bereits vorhandenen Parallelisierungsoption ausgeführt werden, was zum Beispiel dann sinnvoll ist, wenn mehrerer Partitionen zur Änderung anstehen.

4.2 Online Datenredefinition

Der Businesssektor erfordert meist regelmäßige Änderungen der Datenbank, sei es durch quartalsmäßige Abrechnungen oder festgelegte Partitionierungsmaßnahmen, mit der Pflicht einer stetigen Verfügbarkeit der zugrunde liegenden Daten. Oracle sieht für diese Arbeit der Online Schema- und Datenüberarbeitung zwei grundlegende Funktionalitäten vor, die Datenredefinition und die Datenreorganisation.

4.2.1 Redefinition im Überblick

Für die in Oracle umgesetzte Datenredefinition gilt dabei stets der Grundsatz, dass alle Änderungen in eine neu angelegte Tabelle einfließen, während die Alte weiterhin stetig verfügbar ist. Nachdem alle Anpassungen vorgenommen wurden, kann der Datenbankadministrator dann festlegen, wann die endgültige Übernahme der neuen Tabelle erfolgt. Innerhalb dieser online-Änderungsphase stehen ihm dabei die folgenden Funktionen zur Verfügung:

- Modifikationen an den physischen Attributen und Speicherparametern von Tabellen, Erweiterten Queue oder geclusterten Tabellen
- Verschieben einer heap table oder IOT in einen anderen Tablespace
- Erlauben oder Verweigern paralleler Queries
- Erlauben oder Verweigern von Partitionierung
- Reorganisation einer einzelnen Partition
- Erneuerung einer heap table oder IOT, um deren Fragmentierung zu verringern
- Umwandeln von heap tables in IOTs und umgekehrt
- Hinzufügen, Entfernen oder Umbenennen von Spalten in Tabellen
- Umwandeln von LONG oder ONG RAW Spalten in LOB
- Umwandeln von Daten in einer Tabelle

- Erneuern einer Tabelle, die einen abstrakten Datentyp enthält

Für die Reorganisation selbst wird der Oracle Enterprise Manager (OEM) bzw. SQL*Plus empfohlen. Erstgenannter führt graphisch durch die einzelnen Anwendungsschritte, während Letzteres die Funktionalität auf Kommandozeilenebene bietet. Um zum Beispiel eine vollständige Redefinition einer Tabelle durchzuführen, nennt [ODB05a] die folgenden Schritte:

1. eine neue, vorläufige Tabelle mit den gewünschten Attributen erstellen
2. den Redefinitionsprozess starten unter Angabe der Namen des Schemas, der zu reorganisierenden Tabelle, der vorläufigen Tabelle, allen notwendigen Angaben über Mapping der alten Spalten auf die Neuen, dem Typ der Reorganisation (primary key, unique index) und optionalen Parametern über Partitionen
3. wenn nötig, neue Trigger, Constraints, Indexe und Rechte auf die vorläufige Tabelle setzen
4. evtl. schon vorhandene Trigger, Constraints, Indexe und Rechte von der alten Tabelle auf die vorläufige Tabelle kopieren und anpassen
5. optional periodisches synchronisieren und validieren der vorläufigen Tabelle, um evtl. in der Zwischenzeit auf der Originaltabelle in umfangreichem Maße ausgeführte DML Anweisungen anzugleichen und damit den Zeitraum der Finalisierung zu reduzieren
6. Finalisieren der Redefinition, wobei die Originaltabelle so an die vorläufige angepasst wird, dass alle Attribute, Indexe, Trigger, Constraints und Rechte übereinstimmen und aktiviert werden

Bei erfolgreichem Abschluss der Operation werden alle bisherigen Statistiken übernommen und alle geklonten Objekte erhalten ihre Originalbezeichnung zurück. Im Anschluss daran verlieren alle PL/SQL Prozeduren und Cursor, die auf der alten Tabelle definiert waren, ihre Gültigkeit, werden jedoch automatisch neu validiert, sobald sie wieder benötigt werden.
Sollte der Prozess der Reorganisation in einem Stadium fehlschlagen oder hat der Administrator den Wunsch ihn abzubrechen, so werden einfach alle temporär angelegten Logs und Tabellen gelöscht. Danach kann auch die vorläufige Tabelle entfernt werden.

4.2.2 Beispiel: Tabellenredefinition zur Laufzeit

Das folgende Beispiel zeigt die, für die Redefinition der einfachen Tabelle `Artikel` mit den Spalten `Code, Bezeichnung, Preis` in eine nach dem Artikelcode partitionierte Tabelle, notwendigen Schritte, bei gleichzeitiger Preiserhöhung um 16%. Dies alles erfolgt zur Laufzeit auf der Kommandozeile im Schema `s`:

1. vorläufige Tabelle anlegen

```
CREATE TABLE v_artikel ( code NUMBER,
                         bezeichnung VARCHAR(100),
                         preis NUMBER )
    PARTITION BY RANGE( code )
    ( PARTITION code100
        VALUES LESS THAN (100) TABLESPACE tbs1,
      PARTITION code200
        VALUES LESS THAN (200) TABLESPACE tbs2 );
```

2. Redefinition beginnen

```
DBMS_REDEFINITION.START_REDEF_TABLE( 's',
    'artikel', 'v_artikel',
    'code code, bezeichnung bezeichnung, preis*1.16 preis' );
```

3. abhängige Objekte übernehmen

```
DBMS_REDEFINITION.COPY_TABLE_DEPENDENTS( 's',
    'artikel', 'v_artikel',
    1, TRUE, TRUE, TRUE , FALSE, no_errors );
```

4. vorläufige Tabelle synchronisieren (optional)

```
DBMS_REDEFINITION.SYNC_INTERIM_TABLE( 's',
    'artikel', 'v_artikel' );
```

5. Redefinition abschließen

```
DBMS_REDEFINITION.FINISH_REDEF_TABLE( 's',
    'artikel', 'v_artikel');
```

6. vorläufige Tabelle wieder löschen

```
DROP TABLE v_artikel;
```

Nach Ausführung dieser Schritte wurde die Tabelle `Artikel` neu definiert, so dass sie nun alle Attribute der vorläufigen Tabelle `v_Artikel` aufweist.

4.3 Online Datenreorganisation

Zusätzlich zur Datenredefinition bietet Oracle die Möglichkeit zur Datenreorganisation und auch hierfür kann das bekannte ALTER Statement verwendet werden. Als zusätzlicher Parameter wirkt `ONLINE` hier kennzeichnend:

Verschiebung von Tabellen

```
ALTER TABLE tabelle MOVE ONLINE;
```

Indexerstellung

```
CREATE INDEX tabelle.bezeichnungsindex
    ON tabelle(bezeichnung) ONLINE;
```

Indexneustrukturierung

```
ALTER INDEX tabelle.bezeichnungsindex REBUILD ONLINE;
```

Indexzusammenführung

```
ALTER INDEX tabelle.bezeichnungsindex COALESCE;
```

Alle diese Änderungsoperationen unterstützen die Oracle Parallelisierungsfeatures in vollem Umfang.

Speicherfreigabe

Größere Datenbanken mit stark frequentierten Tabellen neigen durch die fortwährenden DML-Operationen (UPDATE, DELETE, ...), die durch die Möglichkeiten des Online-ALTERns noch zusätzlich gefördert werden, über kurz oder lang dazu, den ihnen zur Verfügung gestellten Speicherplatz nicht mehr effizient zu nutzen – man spricht in diesem Zusammenhang von der klassischen Fragmentierung, was in den meisten Fällen zu Performanceverlusten bei Queries führt. Vor allem bei vollständigen Tabellenscans führen die spärlich gefüllten Segmente dazu, dass mehr Blöcke eingelesen werden, als eigentlich nötig wären. Um diese, durch die neuen Online-Funktionalitäten auftretenden „schleichenden" Folgen zu reduzieren, hat Oracle das Feature „Automatic Segment Space Management" geschaffen. Hier kann durch Nachstellen des Schlüsselwortes `SHRINK SPACE` für das ALTERn von Tabellen, Indexe und materialisierten Views die Datenbank dazu veranlasst werden, die übergebenen Objekte zu defragmentieren.
Soll zum Beispiel eine Tabelle als Grundlage dienen, so kann folgender Syntax verwendet werden:

```
ALTER TABLE schema.tabelle SHRINK SPACE
```

Es werden dabei zwei Phasen nacheinander abgearbeitet:

1. Komprimierung. Dabei werden durch verschiedene INSERT und DELETE Statements automatisch alle Daten so weit wie möglich an den Anfang des Segmentes verschoben.

2. Speicherfreigabe. Der in Phase 1 frei gewordene Speicherplatz wird dem Segment entzogen und steht fortan wieder im Tablespace anderen Objekten zur Verfügung.

Wird lediglich die Funktionalität aus Phase 1 gewünscht, so kann dies durch Nachstellen von `COMPACT` erreicht werden:

```
ALTER TABLE schema.tabelle SHRINK SPACE COMPACT
```

Hier zeigt sich deutlich die erreichte Komfortabilität, gegenüber den bisherigen Verfahren (siehe 2).

4.4 Schlussfolgerung

Durch die breite Palette der ONLINE Adjustment Features ist es Oracle möglich, ihrer Datenbank eine Vielzahl an Funktionalitäten bereitzustellen. Dazu gehört unter Anderem das Hinzufügen von Constraints, Tiggern und Partitionen zur Laufzeit, das Erstellen, Erneuern und Zusammenführen von Indexen, das Verschieben indexorientierter Tabellen und die Funktionalität von Datenredefinition und Datenreorganisation. Dies erlaubt es den Administratoren während der Laufzeit sowohl Daten-, wie auch Schemaanpassungen mit größtmöglicher Flexibilität bei maximaler Verfügbarkeit durchzuführen.

Kapitel 5

Microsoft SQL–Server 2005 aka Yukon aka 9.0

Microsoft täte schlecht daran, bei dem Funktionszuwachs der Konkurrenz selbst stillzustehen, bieten Sie doch eigens Tools an, um zum Beispiel einfach von einem Oracle System zu ihrem hauseigenen SQL Server zu migrieren. Als die wesentlichsten Neuerungen hinsichtlich der Schema Anpassung lassen sich die folgenden Punkte nennen:

- Online Index Operations
- XML Schema Evolution

Wobei diese Online-Features ausschließlich der Enterprise Edition vorbehalten sind.

5.1 Systemtables Ade

Bei Microsoft wird ein Schema als Sammlung von Datenbankeinträgen in einem Namespace bezeichnet, wobei jedes Element einen eindeutigen Namen hat. In bisherigen Versionen war es so, dass pro Benutzer genau ein ihm zugeordnetes Schema existierte. So konnte jedes Objekt eines Schemas genau seinem Benutzer zugeordnet werden. All diese Meta Daten wurden im so genannten „Microsoft Repository“ aufbewahrt und verwaltet. Mit Version 2000 wurde dieses Feature dann in „Microsoft Meta Data Services“ umgetauft, um schließlich in der aktuellen Version 2005 wieder ganz entfernt zu werden. Nicht gänzlich, denn eine der wesentlichen Neuerungen in SQL–Server 2005 ist das Abstandnehmen von der klassischen Speicherung der Meta Daten in Systemtables. Microsoft ist nach gründlichen Überlegungen dazu übergegangen, alle diesbezüglichen Daten in flexibler, leicht durchsuchbarer XML zu speichern. Dies macht es möglich, Schemas nun unabhängig von ihren Benutzern zu verwalten und führt zu folgenden, weiteren Vorteilen:

- mehrere Nutzer können sich ein Schema teilen
- Löschen von Nutzern ohne ihre gesamten Objekte umbenennen zu müssen

- die Rechteverwaltung der einzelnen Schemas und Objekte hat sich stark vereinfacht
- Objektnamen bestehen jetzt aus vier Teilen, die zur Adressierung verwendet werden: Server.Datenbank.Schema.Objekt

5.2 SQL–Server 2005 im Überblick

5.2.1 Online Index Operations

In SQL–Server 2005 besteht neben der Erzeugung und dem Löschen die Möglichkeit, Indexe online zu verändern, ohne dabei die Funktionalität von INSERT, UPDATE und DELETE zu behindern. Um dies zu erreichen, behält die Datenbank bei Änderung zwei Kopien des Index – eine Originale, auf der weiterhin alle Operationen ausgeführt werden können und eine Temporäre, die genutzt wird, während der Index erneuert wird. Die Server Engine arbeitet dabei alle währenddessen anlaufenden Änderungen in beiden Versionen ein und nach erfolgreichem Abschluss wird der alte Index gelöscht und durch den Neuen ersetzt. Diese Anpassung wird unterstützt für `CREATE INDEX`, `ALTER INDEX REBUILD`, `ALTER INDEX DISABLE`, `DROP INDEX` und `ALTER TABLE DROP CONSTRAINT` Statements. Der genaue Syntax erschließt sich aus [MSS05a].
Im folgenden Beispiel werden alle Indexe der Beispieldatenbank online erneuert:

```
USE Beispieldatenbank;
GO
ALTER INDEX ALL ON Produktion.Produktname
    REBUILD WITH (ONLINE = ON);
```

In diesem Beispiel wird ein neuer Index über der Postleitzahl der Tabelle Personen erzeugt:

```
CREATE INDEX NeuerIndex ON Personen.Anschrift(PLZ)
    WITH (ONLINE=ON);
```

In beiden Fällen weist der Parameter `ONLINE` die Datenbank an, die Operation zur Laufzeit durchzuführen.

5.2.2 XML Schema Evolution

neuer XML Datentyp

Ein wesentlicher Vorteil von SQL–Server 2005 ist der Support von XML Datentypen, um XML Dokumente in Tabellen zu speichern und der damit verbundene Support von XQuery. Ergänzend dazu (und abweichend vom W3C XML-Standard) hat Microsoft die XML Data Manipulation Language (XML DML) geschaffen, um diese XML Dokumente zur Laufzeit manipulierbar zu machen. Dabei wurde XQuery um dir folgenden Schlüsselwörter erweitert: `insert`, `delete` und `replace value of`. Auch die Indexierung dieses Datentyps ist möglich, was die Performance erheblich steigert, da die einzelnen Dokumente

bei Anfragen nicht jedes Mal zerpflückt werden müssen. Eine Änderung der bestehenden Daten kann sich dann ähnlich dem folgenden Beispiel gestalten:

```
replace value of
    (/Root/Koordinaten/@X)[1]
with
    "100.0"
```

Das Hinzufügen eines oder mehrerer dieser neuen XML Datentypen (ohne zugehörigem Schema) zu einer Tabelle neben eventuell vorhandenen relationalen Attributen gestaltet sich in folgender Art und Weise:

```
CREATE TABLE Tabelle (id INT PRIMARY KEY, doc XML not null)
```

Ähnliches Vorgehen mit vorhandenem Schema `schema`:

```
CREATE TABLE Tabelle (
   id INT PRIMARY KEY,
   doc XML(CONTENT schema))
```

In beiden Fällen werden die XML Datentypen in BLOBs verpackt in der Datenbank abgelegt.

XML Schema Support [MSS05b]

Zum Ablegen der W3C supported XML Schemas bietet SQL Server 2005 die Möglichkeit, diese Meta Daten als so genannte XML Schema Collections zu speichern. Der Begriff Collections wird aus dem Grund verwendet, da es hier möglich ist, mehrere Schemas abzulegen und somit einem XML Dokument, dem diese Collection zugewiesen wurde, mit mehreren Schemas gleichzeitig zu verbinden. In diesem Zusammenhang spricht Microsoft auch von einem „typed XML“, ohne Zuweisung von einem „untyped XML“.
Zur weiteren Veranschaulichung soll der folgende Code dienen:

```
-- Erzeugen einer XML Schema Collection:
CREATE XML SCHEMA COLLECTION MyColl AS '
   <schema
    xmlns="http://www.w3.org/2001/XMLSchema"
    targetNamespace="http://MySchema/test_xml_schema">
      <element name="root" type="string"/>
  </schema>'
-- Ändern einer Collection:
ALTER XML SCHEMA COLLECTION MyColl ADD '
  <schema xmlns="http://www.w3.org/2001/XMLSchema"
         targetNamespace="http://MySchema/test_xml_schema">
     <element name="anotherElement" type="byte"/>
 </schema>'
```

Schemaanpassungen [RYS04]

Um nun das Schema eines XML Datentypen anzupassen, gibt es verschiedene Möglichkeiten:

1. Hinzufügen eines neuen Schemas im Namespace: Dies ist der am wenigsten problematische Weg, da die Daten ihren bisherigen Namespace und ihre vorhandenen Schemas behalten. Dazu ist entweder eine neue Schema Collection zu erstellen und diese einer neu angelegten Spalte zuzuweisen, oder die zu einer bereits vorhandenen Spalte gehörige Schema Collection um das neue Schema zu ergänzen. Sollen auch die zugrunde liegenden XML Daten geändert werden, so hat dies über einen XQuery Ausdruck oder eine XSLT Transformation zu erfolgen.

2. Erweitern eines vorhandenen Schemas: XML Schema Collections erlauben das Hinzufügen neuer Typen, Elemente oder Attribute zu bestehenden Schemas, solange diese neuen Komponenten das vorhandene Schema nicht restriktiver machen und somit eine erneute Validierung auslösen. Dies tritt vor allen beim Einfügen übergeordneter Elemente ein.

3. Austauschen eines vorhandenen Schemas: Soll das gesamte Schema gegen ein neues ausgetauscht werden, so müssen die zugrunde liegenden Daten neu validiert werden. Hierzu ist folgender Ablauf nötig:

   (a) das neue Schema der Schema Collection hinzufügen

   (b) den XML Daten durch `ALTER TABLE` das alte Schema entziehen, diese sind nun „untyped XML“

   (c) optional: Datentransformationen durchführen

   (d) den XML Daten durch `ALTER TABLE` das neue Schema zuweisen, diese sind nun wieder „typed XML“

   (e) optional: das alte Schema löschen

5.3 Schlussfolgerung

Microsoft hat durch die Einführung der XML Schema Evolution ein neues Tor geöffnet, dessen tatsächlicher Nutzen sich erst in den kommenden Monaten zeigen wird. Denn nicht zum ersten Mal wurden vielbeworbene Features in späteren Folgeversionen wieder aus dem Produkt entfernt (vergleiche „English Query“). Der Redmonder Riese sagt von sich selbst, diesen neuen Weg durch umfangreiche Gespräche mit ihren Kunden gefunden zu haben [PAS04], daher ist Microsoft zuversichtlich hinsichtlich der Akzeptanz auf dem Markt.

Kapitel 6

Zusammenfassung, Probleme und Perspektiven

Diese Arbeit hat mögliche Wege der Schema Evolution in aktuellen Datenbanksystemen kurz aufgezeigt und Zusammenhang sowie Unterschiede der führenden kommerziellen Anbieter näher beleuchtet. Alle der betrachteten Systeme bieten grundlegende Funktionalitäten, die es ermöglichen, zur Laufzeit Anpassungen sowohl der Daten, wie auch zugrunde liegenden Strukturen durchzuführen, ohne dabei das komplette System oder Teile davon anhalten zu müssen. Alle Hersteller versprechen weiterhin ihre bisherigen Features zu erweitern und zu verbessern, mit dem Ziel der vollständigen Integration aller möglichen Anpassungen. Durch die kontinuierliche Weiterentwicklung der Systeme verstehen sie sich ausnahmslos auf die Überprüfung syntaktischer Korrektheit. Weiterhin ungelöst bleibt jedoch die semantische Erkennung der bei Evolution teilweise entstehenden Inkonsistenzen, wie sie zum Beispiel bei den zu einer Tabelle gehörigen, durch ein überall mögliches ALTER TABLE ergänzbaren, CHECK Bedingungen auftreten können. Wird hier ein Attribut x mit bestehendem CHECK $x < 50$ um einen neuen CKECK $x > 50$ erweitert, so gibt es noch keine Umsetzung einer erfolgreichen Erkennung dieser Widersprüche. Es wird der Kompetenz des Systempersonals vorbehalten, solche Regeln korrekt umzusetzen, obwohl bereits seit mehreren Jahren verschiedene theoretische Abhandlungen mit Lösungsvorschlägen auf diesem Gebiet angeboten werden [SKN89, GSW96].

Literaturverzeichnis

[WIK05] Wikimedia Foundation Inc.: *Wikipedia Online Encyclopedia*, 2005

[IBM04] IBM Corporation: *Redbook - DB2 UDB for z/OS Version 8 Technical Preview*, 4.3ff, 2004

[MUL04] Craig S. Mullins, *DB2 Developer's Guide, 5th edition*, **SAMS**, 2004

[SKN89] X. Sun, N. Kamel, L. Ni.: *Processing Implications on Queries*, **IEEE Transactions on Software Engineering** 15(10), 1168-1175, 1989

[GSW96] S. Guo, W. Sun, M. Weiss: *Solving Satisfability and Implication Problems in Database Systems*, **ACM Transactions on Database Systems** 21(2), 270-293, 1996

[DBA05] Roger Miller: *www.dbazine.com/db2/db2-mfarticles/emullins-wp2*, 2005

[ODB05a] Oracle Corporation, *Oracle Database 10gR2 Online Data Reorganization & Redefinition An Oracle White Paper*, 2005

[ODB05b] Oracle Corporation, *Oracle Database 10gR2 Administrator's Guide*, 2005

[MSS05a] Microsoft Corporation, *Microsoft SQL Server 2005 CTP June 2005*, 2005

[MSS05b] Microsoft Corporation, *MSDN Library: XML Support in Microsoft SQL Server 2005*, 2005

[RYS04] Michael Rys, *XML Schema evolution in SQL Server 2005*, 2004

[PAS04] PASS, *Interview with Bill Baker @ Microsoft*, 2004