Eigenschaftsbasierte Replikation in mobilen Datenbanksystemen[*]

Hagen Höpfner, Marco Plack

Otto-von-Guericke-Universität Magdeburg
Fakultät für Informatik
Institut für Technische und Betriebliche Informationssysteme
Postfach 4120, D-39106 Magdeburg, Germany
eMail: {*hoepfner* | *plack*}*@iti.cs.uni-magdeburg.de*

18. September 2001

[*]Diese Arbeit wird teilweise von der DFG gefördert (SA 782/3-1).

Hagen Höpfner, Marco Plack:
Eigenschaftsbasierte Replikation in mobilen Datenbanksystemen
Preprint, Fakultät für Informatik,
Otto-von-Guericke-Universität Magdeburg, 2001.

Zusammenfassung

Immer mehr leistungsfähige mobile Endgeräte sind auf dem Markt verfügbar. Durch die zunehmende Leistungsfähigkeit dieser Endgeräte steigt die Zahl vernetzter, mobiler Anwendungen rapide, die vor kurzer Zeit noch fest in der Hand der klassischen Client-Server Anwendungen zu finden waren. Auf Grund der Einschränkungen hinsichtlich verfügbarer Speicherkapazität und Bandbreite von drahtlosen Kommunikationsnetzen werden neue Verfahren benötigt, die eine automatische Reduktion von Daten bei der Übertragung zwischen Server und mobile Client ermöglichen.

Bisherige Lösungen haben den Nachteil das die Auswahl der abonnierten Daten von Hand durchgeführt werden muss. In diesem Preprint wird ein Verfahren vorgestellt, dass eine automatische Auswahl der optimal zu übertragenen Datenmenge unterstützt und damit zur Realisierung neuer, bisher nicht umsetzbarer Anwendungen für den Sektor mobiler Endgeräte beiträgt. Basis dieses Verfahrens ist die Erweiterung der Daten in Datenbanken um Informationen, die zusätzliche Eigenschaften des Datenbestandes beschreiben und für eine automatische Replikatoptimierung eingesetzt werden können. Neben diesem neuen Ansatz werden vorhandene Techniken vorgestellt die auf einem ähnlichen Ansatz basieren und kurz auf beispielhafte Anwendungen des neuen Verfahrens eingegangen.

Inhaltsverzeichnis

Kapitel 1

Einleitung

Aufgrund der rasanten technischen Entwicklung auf dem Gebiet der mobilen Hardware nimmt die Verbreitung von mobilen Endgeräten in der Industrie und in Haushalten immer mehr zu. Durch die gesteigerte Leistungsfähigkeit mobiler Geräte werden immer neue Anwendungen erschlossen, die bisher mit klassischen, festnetzgebundenen Client-Server-Systemen realisiert worden sind und jetzt mit mobilen Endgeräten flexibler eingesetzt werden können. Aufgrund der Mobilität der Clients lassen sich neue Anforderungen an die Kommunikation zwischen Client und Server definieren:

- Die Struktur von Kommunikationsnetzen, die auch mobile Endgeräte enthalten, ist nicht statisch. Zu einem bestimmten Zeitpunkt ist nicht vorhersagbar, welche mobilen Endgeräte für eine Kommunikation zur Verfügung stehen.
- Die Bandbreite eines Kommunikationskanals ist nicht statisch. Beispielsweise können mobile Geräte im Gelände oft nur schmalbandige Kommunikationskanäle nutzen.
- Die Hardware von mobilen Endgeräte ist oft nicht so leistungsfähig wie die stationärer Geräte.

Durch diese Einschränkungen, die durch limitierte Ressourcen von mobilen Geräten verursacht werden, ist es oft notwendig die Datenmenge, die von einem mobilen Endgerät im Vergleich zu einem stationären Endgerät verwaltet und verarbeitet werden kann, einzuschränken. Für Datenbanken, die mit mobilen Endgeräten kommunizieren, bedeutet dies, dass effiziente Techniken zur Synchronisation bzw. Replikation von Daten erforderlich sind, die Methoden zur Datenreduktion bereits beinhalten. Bisherige Ansätze zur Einschränkung der Datenmenge basieren auf der manuellen Spezifikation der zu replizierenden Daten. Auch in Hinsicht auf Anforderungen aus dem Bereich von mobilen Kommunikationsnetzen, wo Verfahren zur positionsabhängigen Replikation benötigt werden, sind Verfahren zur manuellen Vorauswahl der zu replizierenden Daten nicht mehr zufriedenstellend. Verfahren, die eine positionsabhängige Replikation ermöglichen, erweitern den Datenbestand um geographische Positionsdaten. Diese zusätzlichen Informationen werden zur Laufzeit ausgenutzt um relevante Daten zum Replikationszeitpunkt zu bestimmen.

In diesem Papier werden Ansätze vorgestellt, die zur automatischen Ermittlung des optimalen Replikats eingesetzt werden können. Diese Ansätze werden dann zu einem Verfahren zur ereignisbasierten Replikation verallgemeinert.

Auf der Basis des vorgestellten Verfahrens lässt sich eine Middelware-Plattform für mobile Anwendungen entwickeln, auf deren Basis sich verschiedenste mobile Informationssysteme entwickeln lassen. Im folgenden werden zwei mögliche Anwendungen kurz vorgestellt:

Am Markt verfügbare Navigationsysteme verfügen über genaues Wissen über den Verlauf von Straßen. Fügt man zu diesen Daten Informationen über Plätze mit erhöhtem Unfallrisiko, wie z.B. Schulen, Kindergärten oder auch Stauenden hinzu, kann man Fahrer von Personenkraftfahrzeugen effektiv auf erhöhtes Unfallrisiko hinweisen. Aufgrund der limitierten Speicherkapazität der verfügbaren Systeme und der Dynamik in den benötigten Informationen (z.B. Stauinformationen, Kurzzeitbaustellen) ist es erforderlich, diese Daten dynamisch und positionsabhängig an Fahrzeuge zu verbreiten. Ein Ansatz der das positionsabhängige Replizieren von Feindaten ermöglicht, würde die Realisierung moderner Fahrerinformationssysteme unterstützen. Auf der Basis eines solchen Systems lassen sich dann auch weitere Dienste, wie z.B. dynamische Verkehrsführung oder Navigation zu touristischen Sehenswürdigkeiten realisieren.

Wartungsmitarbeiter im Außendienst werden häufig dynamisch neue Einsatzorte in ihrer Reichweite zugeordnet. Leider ist es oft für den Außendienstmitarbeiter nicht möglich, alle Wartungsunterlagen oder Baupläne für alle betreute Produkte mitzuführen. Ein kombiniertes System aus automatischer Replikation und Objekterkennung kann eingesetzt werden, um benötigte Unterlagen auf das mobile System des Mitarbeiters zu replizieren, ohne dass eine Interaktion des Technikers erforderlich ist.

Realisiert werden kann ein solches Verfahren durch ein System, dass es erlaubt, den Datenbestand bzw. Teile des Datenbestandes (Fragmente) um zusätzliche Informationen (Extensionen) zu erweitern. Diese Extensionen können dann für eine automatische Bestimmung des optimalen Replikats zur Laufzeit des Systems ausgenutzt werden. Basierend auf der in [PS98] vorgestellten Klassifikation von Knoten in mobilen Datenbanksystemen ergibt sich das in Abbildung 1.1 dargestellte grundlegende Szenario, welches sich für den in dieser Arbeit diskutierten Ansatz zum Einsatz kommt.

Bei der Synchronisation zwischen dem mobilen Knoten und der Basisstation überträgt der mobile Knoten zuerst die Werte der ihm bekannten Extensionen an den Basisknoten. Die Vorauswahlschicht des Basisknotens wertet die vom mobilen Knoten empfangenen Informationen aus und fordert unter Beachtung der gespeicherten Fragmentdefinitionen

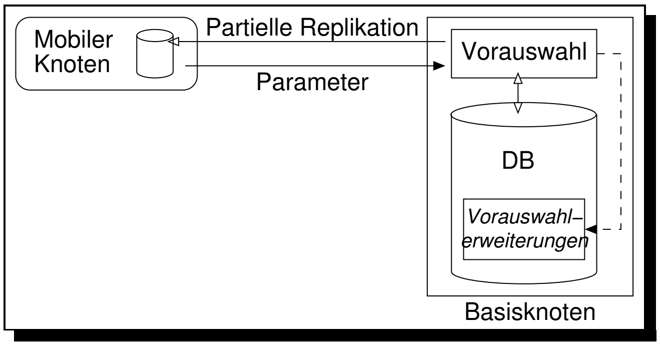

Abbildung 1.1: Szenario

nur die notwendigen Daten von der Datenbank an und überträgt diese zurück an den mobilen Knoten.

Da eine Extension sich auf einen oder mehrere Ausschnitt(e) einer oder mehrerer Relation(en) beziehen kann (vgl. Abbildung 1.2), kann die Speicherung der Extension in einer Datenbank nicht generell als zusätzliches Attribut der betroffenen Relationen erfolgen. Ein Vorteil dieses Ansatzes ist die Möglichkeit, bereits bestehende Datenbanken

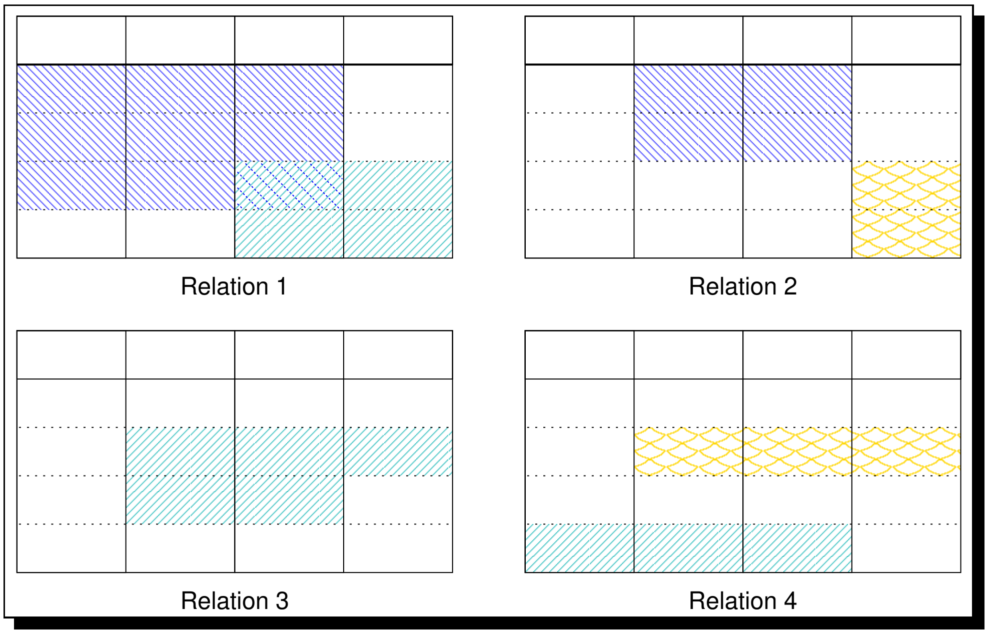

Abbildung 1.2: Beispiele für Fragmentierung von Relationen

einfach um die notwendigen Fragmentinformationen zu erweitern. Dazu wird der Teil der bestehenden Datenbank, welcher bezüglich einer Extension (z.B. geographisch, zeitlich, Zugehörigkeit zu einer Gruppe) relevante Informationen enthält, durch eine

Fragmentierungsfunktion in Form einer Datenbankanfrage (z.B. in SQL) beschrieben und durch eine instantiierte Extension einem expliziten Bereich dieser Extension (z.B. gehört zu Gruppe A, relevant für Dienstag) zugeordnet. Ausschnitte einer Relation die durch eine solche Fragmentierungsfunktion beschrieben sind werden als Fragmente bezeichnet.

Abbildung 1.2 zeigt, das Fragmente auch relationsübergreifend und überlappend sein können. Relation 3 in Abbildung 1.2 zeigt ein Fragment, welches aus zwei Tupeln mit zwei bzw. drei Attributen besteht. Ein solches Fragment kann beispielsweise unter Verwendung der `UNION`-Anweisung von SQL realisiert werden, wenn diese auf die Rückgabe zweier `SELECT`-Anweisungen angewendet wird. Das dritte (fehlende) Attribut des zweielementigen Tupels wird hierbei mit NULL aufgefüllt. Da es sich bei der Definition von Fragmentierungsfunktionen lediglich um die Anwendung einer Anfragesprache handelt, soll dieses Thema an dieser Stelle nicht weiter vertieft werden. Die Möglichkeiten der Definition mit Hilfe der Anfragesprache SQL können in [KK01] nachgelesen werden.

Kapitel 2

Stand der Technik

In diesem Kapitel werden bekannte Replikationsverfahren vorgestellt und Vor- und Nachteile der einzelnen Ansätze erläutert. Insbesondere werden die Verfahren hinsichtlich ihrer Eignung für eine automatischen Einschränkung des Datenbestandes bei der Replikation untersucht.

2.1 Replikation mit Hilfe von Sichten am Beispiel Oracle

Ein klassischer Ansatz zum „Ausblenden“ von nicht benötigten Information ist die Definition von Sichten [HS00]. Existierende Datenbankmanagementsysteme, wie z.B. DB2 [Cha96], Sybase [Goo92] Informix [Pet97] oder Oracle [CL90] unterstützen inzwischen Replikation zwischen verschiedenen Datenbankinstanzen mit Hilfe von Sichten (Snapshots). Am Beispiel von Oracle werden im folgenden die Möglichkeiten dieser Form der Replikation beschrieben. Neben der Multi-Master Replikation, die eine Form der Read One Write All (ROWA) Replikation [Dad96] darstellt unterstützt Oracle die Replikation mit Hilfe von ReadOnly- bzw. Updateable-Snapshots. Bei diesen Formen der Replikation wird auf dem Client eine Sicht auf die Originaldaten des Primärknotens erzeugt und gleichzeitig angegeben in welchen Zeitintervallen der Snapshot aktualisiert werden soll. Die genaue Syntax und Vorgehensweise zum Erstellen eines Snapshots kann in [Dye99] nachgelesen werden. Der Nachteil dieser Verfahren liegt darin, das der Client zum Aktualisierungszeitpunkt erreichbar sein muss. Es wird davon ausgegangen, dass alle Knoten zu jedem Zeitpunkt erreichbar sind, was in einem mobilen Umfeld jedoch nicht gewährleistet werden kann. Ein weiterer Nachteil ist, dass die Ausschnitte aus dem Datenbestand des Primärknotens bereits zum Initialisierungszeitpunkt definiert werden müssen. was zur Folge hat, dass eine nachträgliche Anpassung des zu replizierenden Ausschnittes aufgrund geänderter Ausgangsbedingungen beim Client nicht möglich ist. Ein Vorteil bei Verwendung von Views zur Replikation, in Hinsicht auf mobile Systeme, ist das der Primärknoten den zu replizierenden Ausschnitt kennt und somit zu jedem Zeitpunkt weiß, welche Änderungen noch an den Client zu übertragen sind, was zu einer Reduktion des Übertragunsvolumens ausgenutzt werden kann.

2.2 Einsatz von materialisierten Sichten

In [LLS00] wird die Verwendung von materialisierten Sichten vorgeschlagen, welche laut [ZHK96] u.a. bei unzuverlässigen Netzwerkverbindungen besser geeignet sind als virtuelle Sichten.

Das in [LLS00] vorgestellte Verfahren basiert auf dem folgenden Szenario. Es existiert ein Datenbankserver DB, welcher die Basisrelationen R_i $(i, n \in \mathbb{N}, 1 \leq i \leq n)$ verwaltet. Jede Relation R_i besteht aus einer Menge von Attributen $A_i = \{A_{i_1}, A_{i_2}, \cdots, A_{i_{|A_i|}}\}$. Gelöschte Tupel bleiben in der Datenbank gespeichert. Dazu wird die Datenbank DB in die beiden Teile DB^+ und DB^- geteilt wird. DB^+ enthält den aktuellen Datenbestand. In DB^- werden gelöschte Tupel protokolliert. Es wir gefordert, dass $DB^+ \cup DB^- = DB$ gilt. DB^+ und DB^- sind äquivalent zu DB als $DB^+ = \{R_1^+, R_2^+, \cdots, R_n^+\}$ bzw. als $DB^- = \{R_1^-, R_2^-, \cdots, R_n^-\}$ definiert. Die Relationen in DB^+ werden um einen *Erzeugungszeitstempel* $c\tau_i$ erweitert. Demzufolge gilt für die Attribute der Relation R_i^+: $A_i^+ = A_i \cup \{c\tau_i\}$. Daneben erhält A_i^- als zusätzliches Attribut einen *Löschzeitstempel* $d\tau_i$. Folglich gilt $A_i^- = A_i^+ \cup \{d\tau_i\}$ bzw. $A_i^- = A_i \cup \{c\tau_i, d\tau_i\}$.

In diesem Szenario sind die Operation *Tupel-Einfügen*, *Tupel-Löschen* und *Tupel-Ändern* zugelassen. Hierbei wird das Ändern eines Tupels durch das Löschen der alten Version des Tupels und das Einfügen der neuen Version des Tupels realisiert. Damit ergibt sich ein zeitabhängiger Datenbankzustand $R_i(t)$, wobei t den Zeitpunkt beschreibt. Es wird festgelegt, dass R_i der aktuelle Datenbankzustand ist. Weiterhin werden die Attributwerte $A_{i,j} \subseteq A_i$ eines Tupels $r_k \in R_i$ mit $r_k[A_{i,j}]$ beschrieben. Die einzelnen Operationen werden beschrieben durch:

- Tupel in R_i einfügen: r_k^{insert} mit dem Erzeugungsstempel $r_k^{+,insert}[c\tau_i] = t_{insert}$ von $r_k^{+,insert}$ wobei t_{insert} der Zeitpunkt des Einfügens des Tupels ist

- Tupel in R_i löschen: r_k^{delete} mit dem Löschstempel $r_k^{-,delete}[d\tau_i] = t_{delete}$, wobei $r_k^{+,delete}$ aus R_i^+ entfernt und als $r_k^{-,delete}$ in R_i^- zum Zeitpunkt t_{delete} eingefügt wird

- Tupel $r_k \in R_i$ ändern in r_k^*: äquivalent, da zuerst r_k^+ in R_i^+ und als r_k^- in R_i^- eingefügt wird (siehe löschen) und dann $r_k^{+,*}$ in R^+ eingefügt wird. (siehe einfügen)

Für die Definition von Fragmenten müssen bei diesem Ansatz, welcher eine lokationsabhängige Replikation zulässt, die notwendigen Informationen bereits in den Relationen der Datenbank enthalten sein.

Die in diesem Kapitel beschriebenen Verfahren zeigen, dass existierende Verfahren in Hinblick auf die automatische Reduktion auf relevante Daten bei der Replikation nicht geeignet sind. Zum einen fehlt die nachträgliche Anpassung des zu replizierenden Datenausschnittes an geänderte Eigenschaften des Clients und zum anderen kann der Datenbestand nicht um zusätzliche Eigenschaften zur Bestimmung des optimalen Replikats erweitert werden. Im folgenden Kapitel wird ein Verfahren vorgestellt, dass diese Nachteile nicht hat.

Kapitel 3

Basisszenario

In diesem Kapitel werden die bereits in der Einleitung genannten Begriffe (vgl. Abbildung 3.1) detailliert erläutert und in Abschnitt 3.1 formal definiert. Dazu wird im folgenden zuerst allgemein das Verfahren der eigenschaftsbasierten Replikation, vorgestellt. Wie bereits erwähnt, werden Fragmente bei der eigenschaftsbasierten Replikation

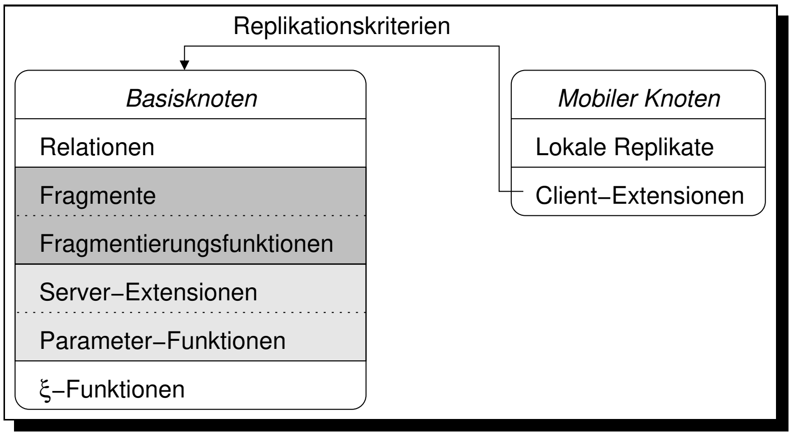

Abbildung 3.1: Übersicht der eingeführten Begriffe

um Extensionen v erweitert. Hierbei wird zwischen Server-Extensionen $v^{se}_{name} \in \mathcal{V}_{se}$ und Client-Extensionen $v^{ce}_{name} \in \mathcal{V}_{ce}$ unterschieden. Bei beiden handelt es sich um Vektoren, welche die mit *name* identifizierte Eigenschaft beschreiben (z.B. zweidimensionale geographische Koordinaten $v_{geogcoord}(x, y)\ x, y \in \mathbb{R}$). Die Fragmente auf dem Server werden folglich um eine Menge von Server-Extensionen erweitert. Client-Extensionen kommen erst bei der eigentlichen Replikation zum Einsatz und dienen dann als „Vergleichswert" um anhand der gespeicherten Server-Extensionen die Auswahl relevanter Fragmente zu ermöglichen. Der Unterschied zwischen Server- und Client-Extensionen besteht in der Mächtigkeit der Parameter (im letzten Beispiel x und y). Serverseitig sind für die Parameter Funktionen zulässig, welche die entsprechenden Parameterwerte „berechnen".

Durch diese Parameter-Funktionen können somit bereits in der Datenbank enthaltene Informationen genutzt werden, was eine redundante Speicherung dieser Informationen überflüssig macht. In dem Fall, dass keine solchen Informationen vorhanden sind, die für eine Berechnung der Extensionswerte genutzt werden können, können die Fragmente mittels nullstelliger Funktionen, also Konstanten, explizit angegeben werden. Client-Extensionen werden ohne vorheriges Wissen über die auf dem Server vorhandenen Daten in Form von Replikationskriterien vom mobilen Client an den Server übertragen. Aus diesem Grund müssen die Parameter der Client-Extensionen nullstellige Funktionen sein.

Die Erweiterung von Client-Extensionen zu Replikationskriteria ist notwendig, da nicht nur „Punktanfragen“ realisiert werden sollen, sondern auch „Bereichsanfragen“ möglich sein müssen. Folglich umfassen Replikationskriteria neben einer Client-Extension auch Grenzwerte für eine minimal geforderte bzw. eine maximal zulässige Abweichung der Server-Extension eines Fragmentes von der Client-Extension. Die Verwendung einer maximal zulässigen Abweichung ist intuitiv. Ein minimal geforderter Abstand ist insbesondere bei geographischen Extensionen dadurch zu rechtfertigen, dass es somit möglich ist, zusätzliche Daten zu bereits vorhandenen Informationen zu replizieren. Wenn beispielsweise ein mobiler Client bereits Informationen über seinen aktuellen Standort besitzt, ist es offensichtlich sinnvoller, diese nicht erneut zu replizieren, sondern mit einem minimal geforderten Abstand lediglich auf Daten mit einem Bezug zu einem gewissen Umkreis zurückzugreifen.

Bei der Bestimmung der zu replizierenden Fragmente kommt eine von der Extension abhängige ξ-Funktion zum Einsatz, welche den „Abstand“ zwischen einer Client- und einer Server-Extension berechnet. Ist dieser Wert größer-gleich der minimal geforderte und kleiner-gleich der maximal zulässigen Abweichung, wird das entsprechende Fragment repliziert.

3.1 Formale Definition

In diesem Abschnitt werden die verwendeten Begriffe formal definiert. Die Definitionen basieren auf der in [HS00] vorgestellten Formalisierung des Relationenmodells.

Definition 1: Attribut Ein Attribut $A \in \mathcal{U}$ mit dem Wertebereich $D = dom(A)$ ist definiert als Element der nicht-leeren, endlichen Menge $\mathcal{U}$ (*Universum* der Attribute).

Definition 2: Relationenschema Ein Relationenschema ist definiert als Menge $R \subseteq \mathcal{U}$.

Definition 3: Relation und Tupel Ein Tupel t ist definiert als Abbildung $t : R \rightarrow \bigcup_{i=1}^{m} D_i$. Eine Relation ist eine endliche Menge von Tupeln.

Definition 4: Parameter-Funktion Eine Parameter-Funktion ψ ist eine Abbildungen der Form $\psi = r(R) \rightarrow x$ wobei x einem zulässigen SQL-Datentyp genügen muss. Ψ ist die Menge aller Parameter-Funktionen.

Definition 5: Server-Extension Eine Server-Extension ist definiert durch den Vektor $v^{se}_{name} = (d_0, \cdots, d_i)$ mit dem identifizierenden Namen ($name$) und d_i ($i \in \mathbb{N}$) Parameter-Funktionen zur Berechnung der Extensions-Parameter-Werte. $\mathcal{V}_{se}$ ist die Menge aller Server-Extensionen.

Definition 6: Client-Extension Von einer Client-Extension v^{ce}_{name} wird neben den Eigenschaften der Server-Extension gefordert, dass die Funktionen d_i nullstellig und somit konstant sind. $\mathcal{V}_{ce}$ ist die Menge aller Client-Extensionen.

Definition 7: Fragmentierungsfunktion Die Fragmentierungsfunktion $f = r(R) \rightarrow r'(R)$ mit $r'(R) \subseteq r(R)$ ist definiert als $f = \pi_{a_1,\cdots,a_n}\sigma_{c_1,\cdots,c_m}(r(R))$ mit $n, m \in \mathbb{N}$.

Definition 8: Fragment Ein Fragment einer Datenbank ist definiert als $\mathcal{F} = (f, V)$ mit der Fragmentierungsfunktion f und einer Menge von Server-Extensionen $V = \{v_0, \cdots, v_n\}$ mit $n \in \mathbb{N}$ und $v_0, \cdots, v_n \in \mathcal{V}_{se}$.

Definition 9: ξ-Funktion Die ξ-Funktion $\xi_a(v_{a_1}, v_{a_2})$ ist die Abbildung $\mathcal{V}_{se} \times \mathcal{V}_{ce} \rightarrow \mathbb{R}$ von je einer Server-Extension $v_{a_1} \in \mathcal{V}_{se}$ und einer Client-Extension $v_{a_2} \in \mathcal{V}_{ce}$ auf eine reelle Zahl d welche den „Abstand“ zwischen v_{a_1} und v_{a_2} beschreibt. Ξ ist die Menge aller ξ-Funktionen.

Definition 10: Replikationskriterium Ein Replikationskriterium ist definiert durch das Tripel $rk = (v', \Delta^{min}, \Delta^{max})$ mit einer Client-Extension $v' \in \mathcal{V}_{\rceil\rceil}$ und den Grenzwerten $\Delta^{max}, \Delta^{min} \in \mathbb{R}$ der Abweichung von dieser. Hierbei muss $\Delta^{max} \geq \Delta^{min}$ gelten.

3.2 Beispiele für Extensionen und ξ-Funktionen

Der hier vorgestellte generelle Ansatz zur Einschränkung zu replizierender Daten ist äußerst flexible und ermöglicht die Fragmentierung der Datenbank anhand verschiedenster Eigenschaften. In diesem Abschnitt werden einige Beispiele für Extensionen und den zugehörenden ξ-Funktionen vorgestellt. Dabei wird zum einen auf die klassischen Kriterien Zeit (Abschnitt 3.2.1) und Ort (Abschnitt 3.2.2) und zum anderen auf die Gruppenzugehörigkeit (Abschnitt 3.2.3) eingegangen. Neben diesen drei Einschränkungsmöglichkeiten sind noch weitere, wie z.B. Hirarchieebene, Temperatur oder Gaskonzentration denkbar. Dieses weiteren Einschränkungsmöglichkeiten stellen jedoch nur weitere Formen von Extensionen dar und werden im weiteren nicht näher erläutert.

3.2.1 Zeit

Zeit ist ein häufig genutzter Parameter um einen Datenbereich zu einzuschränken. Hier ist die Abbildung des Faktors Zeit in der Form eines kontinuierlichen Intervalls $[\Delta_{Zeit}^{max}, \Delta_{Zeit}^{min}]$ möglich. Daraus resultieren die folgenden Fälle:

1. $v_{Zeit} = (a)$, $\Delta_{Zeit}^{min} = b$, $\Delta_{Zeit}^{max} = c$ (a, b, c sind Zeitwerte)
   Hierbei werden nur Daten mit einem Bezug zum Zeitintervall $[a+b, a+c]$ berücksichtigt. Tabelle 3.1 stellt die hierbei auftretenden Spezialfälle dar.

2. $v_{Zeit} = (a)$ $\Delta_{Zeit}^{min} = b = -\infty$, $\Delta_{Zeit}^{max} = c$
   Hierbei werden nur Daten mit Bezug zu einem Zeitpunkt bis $a+c$ berücksichtigt. (Siehe Tabelle 3.2)

3. $v_{Zeit} = (a)$ $\Delta_{Zeit}^{max} = c = \infty$, $\Delta_{Zeit}^{min} = b$
   Hierbei werden nur Daten mit Bezug zu einem Zeitpunkt ab $a+b$ berücksichtigt. (Siehe Tabelle 3.2)

b	c	Bedeutung
0	0	Fragmente mit Bezug zum Zeitpunkt a
0	> 0	Fragmente mit Bezug zum Zeitpunkt zwischen a und c
< 0	0	Fragmente mit Bezug zum Zeitpunkt zwischen b und a in der Vergangenheit
< 0	> 0	Fragmente mit Bezug zum Zeitpunkt zwischen b und c
> 0	> 0	Fragmente mit Bezug zum zukünftigen Zeitpunkt zwischen $a+b$ und $a+c$
< 0	< 0	Fragmente mit Bezug zum Zeitpunkt zwischen $a - \|b\|$ und $a - \|c\|$ in der Vergangenheit
$\neq 0$	$= b$	Fragmente mit Bezug zum Zeitpunkt b (a nicht notwendig)

Tabelle 3.1: Ausprägungen der Zeit-Extension

Wie in Abschnitt 3 beschrieben, muss für jede Extension eine spezielle ξ-Funktion definiert werden. Für zwei Zeit-Extensionen $v_{Zeit_1} = (a)$ und $v_{Zeit_2} = (b)$ kann diese Funktion mit $\xi_{Zeit}(v_{Zeit_1}, v_{Zeit_2}) = a - b$ festgelegt werden.

3.2.2 Geographische Koordinaten

Verfahren zur Bestimmung von Positionsinformationen mobiler Endgeräte, wie z.B. GPS [Kap96], DGPS [Hur93] oder GLONASS [GMK+98] sind weit verbreitet und wurden detailliert in Publikationen auf ihre Anwendbarkeit in den verschiedensten Bereichen mobiler Informationssysteme betrachtet (z.B. [IN99]). Daher werden als zweites

b	c	Bedeutung
$-\infty$	0	Alles mit Zeitbezug bis a
$-\infty$	> 0	Alles mit Zeitbezug bis c (a eingeschlossen)
$-\infty$	< 0	Alles mit Zeitbezug bis c (a ausgeschlossen, weil $c < (a + c)$)
0	∞	Alles mit Zeitbezug ab a
< 0	∞	Alles mit Zeitbezug ab b (a eingeschlossen)
> 0	∞	Alles mit Zeitbezug ab b (a ausgeschlossen, weil $(b + a) > a$)
$-\infty$	∞	Alles (Zeit-Extension ignorieren)

Tabelle 3.2: Zusätzliche Ausprägungen der Zeit-Extension

Extensions-Beispiel geographische Koordinaten aufgeführt. Unter der Voraussetzung, dass diese Extension der Form[1] $v_{geogcoord} = (x, y)$ entspricht, kann es zur Vorauswahl mit $\Delta^{min}_{geogcoord} = b$, $\Delta^{max}_{geogcoord} = c$ eingesetzt werden. Die daraus resultierenden Fälle der Datenbeschränkung, sind in Abbildung 3.2 dargestellt. Die dritte, in Abbildung 3.2 illustrierte, Variante kann genutzt werden, um eine Erweiterung des Fokusses zu realisieren. Hierbei wird mit $\Delta^{min}_{geogcoord}$ der Datenbereich beschrieben, der bereits auf dem mobilen Knoten vorhanden ist. Es werden folglich nur Daten repliziert, die einen Bezug zu der Region $\Delta^{max}_{geogcoord} - \Delta^{min}_{geogcoord}$ haben. Die ξ-Funktion zwischen

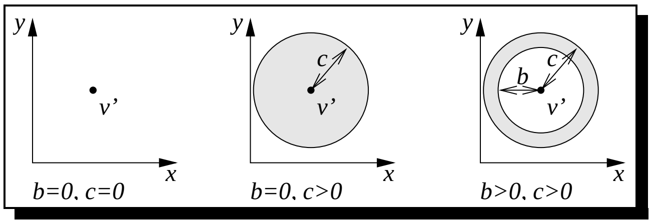

Abbildung 3.2: Ausprägungen der Geographische-Koordinaten-Extension

zwei geographischen Koordinaten $v_{geogcoord_1}(x_1, y_1)$ und $v_{geogcoord_2}(x_2, y_2)$ ist offensichtlich $\xi_{geogcoord}(v_{geogcoord_1}, v_{geogcoord_2}) = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$.

3.2.3 Zugehörigkeit zu einer Gruppe

Neben den in den Abschnitten 3.2.1 und 3.2.2 vorgestellten klassischen Eigenschaften, die zur Beschränkung einer Datenmenge herangezogen werden, bietet der in diesem Preprint diskutierte Vorauswahl-Ansatz auch die Möglichkeit, Datenobjekte bzw. Fragmente aufgrund ihres Bezuges zu einer bestimmten Gruppe zu klassifizieren. Beispiele hierfür sind:

[1]In diesem Beispiel wird der Einfachheit halber die Höhe ignoriert.

- Fragment fasst Räume zusammen, die sich alle in ein und demselben Gebäude befinden $v_{gebäude} = (Gebäude)$.
- Fragment fasst Kunden zusammen, die eine Lebensversicherung abgeschlossen haben $v_{versicherung} = (Abschluß)$
- Fragment fasst Teile zusammen, die in einer gemeinsamen Maschine verbaut sind $v_{maschine} = (Maschinennummer)$

Offensichtlich ist es bei einer solchen Gruppenzugehörigkeit nicht sinnvoll, Abweichungen zuzulassen. Das heißt, dass die entsprechende ξ-Funktionen boolsch sind und bei Erfüllung den Wert 1 und bei Nichterfüllung den Wert -1 liefern.

Angenommen ein Client will alle Daten zu Räumen des Gebäudes 03 replizieren und überträgt dazu eine Replikationskriterium $rk = ((03), 0, 1)$. Die darin enthaltene Client-Extension $v^{ce}_{gebäude} = (03)$ wird auf dem Server mittels der $\xi_{gebäude}$-Funktion mit den dort gespeicherten Server-Extensionen verglichen. Diese Funktion ist für eine Client-Extension $v^{ce}_{gebäude} = (g_1)$ und eine Server-Extension $v^{se}_{gebäude} = (g_2)$ mit

$$\xi_{gebäude}(v^{se}_{gebäude}, v^{ce}_{gebäude}) = \begin{cases} 1 & \text{falls } g_1 = g_2 \\ -1 & \text{sonst} \end{cases}$$

gegeben. Da gefordert ist, dass $\Delta^{min}_a \leq \xi(v^{se}_a, v^{ce}_a) \leq \Delta^{max}_a$ gilt, wenn ein Fragment, welches durch v^{se}_a gekennzeichnet ist, repliziert werden soll, und dies hier aufgrund $\Delta^{min}_{gebäude} = 0$, $\Delta^{max}_{gebäude} = 1$ und $\xi_{gebäude} = \{-1, 1\}$ nur gilt, wenn das Fragment zu dem gesuchten Gebäude gehört, werden nur die gewünschten Daten repliziert.

Kapitel 4

Replikationsprotokoll

Nach der Definition der Fragmente umfasst die eigenschaftsbasierte Replikation im wesentlichen zwei zentralen Punkte. Zum einen müssen Daten unter Beachtung der Eigenschaften der mobilen Clients berechnet und an diese übertragen werden, zum anderen müssen Änderungen an die betroffenen Replikate propagiert werden. In diesem Abschnitt werden zwei Algorithmen vorgestellt, welche diese beiden Aufgaben erfüllen.

4.1 Vorauswahl-Algorithmus

Basierend auf den Definitionen in Abschnitt 3.1 ist Algorithmus 1 zur Preselektion der zu replizierenden Daten gegeben. Er wird auf der Basisstation ausgeführt. Als Eingabe erhält er eine Menge von Replikationskriterien, die Fragmentdefinitionen und die Datenbank. Die Ausgabe des Algorithmuses ist die Menge der Fragmenten, welche mit den Replikationskriterien korrespondieren.

Algorithm 1: *Vorauswahl-Algorithm*

ENSURE: *// INPUTS: fragments, set of replication criterions*
ENSURE: *RETURN*=$\emptyset$; *// reset return set*

for each replication criterion $rk_i = (v', \Delta^{max}, \Delta^{min})$ **do**
  for each fragment $\mathcal{F}_j = (f, V)$ **do**
    if $\exists(v_n \in V | v_n$ is compatible to $v')$ **then**
      if $\Delta^{min} \leq \xi_v(v_n, v') \leq \Delta^{max}$ **then** *RETURN*=$\mathcal{F}_j \cup$ *RETURN*
  done *//end of inner loop*
done *//end of outer loop*
return(*RETURN*);

In der äußeren Schleife werden alle Replikationskriterien durchlaufen. Für jedes dieser Replikationskriterien wird in der inneren Schleife untersucht, mit welchen der gespeicherten Fragmente sie korrespondieren. Das heißt, dass mit Hilfe der entsprechenden

ξ-Funktion der „Abstand“ des jeweiligen Replikationskriteriums von dem Fragment bestimmt wird. Liegt dieser innerhalb der zugelassenen Grenzen, dann wird das Fragment in die Menge der zu replizierenden Fragmente übernommen.

4.2 Kopie-Update-Strategie

Eine zentrale Aufgabe von Replikationstechniken ist laut [Dad96] die *Kopie-Update-Strategie*. Sie wird eingesetzt, um Änderungen des Datenbestandes an die betroffenen Knoten des Netzes zu propagieren. Unter Beachtung des hier vorliegenden Primary-Copy-Ansatzes, d.h. dass Updates nur auf dem Server realisiert werden können, wird in diesem Abschnitt ein Algorithmus vorgestellt, welcher zur Aktualisierung der Replikate auf den mobilen Endgeräten eingesetzt wird. Dieser Algorithmus 2 wird serverseitig ausgeführt.

Das Ziel dieses Ansatzes besteht darin, dass die mobilen Knoten möglichst aktuelle Daten zur Verfügung haben. Um dies zu realisieren, werden die folgenden Informationen über die mobilen Knoten und deren Replikationsverhalten auf dem Server gespeichert:

1. ID der mobilen Knotens
2. Replikationskriteria, welche bei der letzten Replikation genutzt wurden

Tritt eine Änderung des Datenbestandes ein, überprüft der Server anhand der gespeicherten Replikationskriteria, welche mobilen Clients von dieser Änderung betroffen sind und versucht diese unter Verwendung der entsprechenden IDs zu updaten. Dazu fordert der Server die jeweils aktuellen Replikationskriteria von den mobilen Knoten an und prüft, ob sich diese von den gespeicherten unterscheiden. Werden keine Unterschiede festgestellt, müssen nur die geänderten Daten neu repliziert werden. Haben sich die Replikationskriteria des mobilen Knotens geändert, so muss neu repliziert werden. Hierbei werden die getätigten Änderungen automatisch berücksichtigt.

Im dem Fall, dass nicht alle mobilen Knoten erreichbar sind werden die entsprechenden Replikate beim nächsten standardmäßigen Synchronisieren der betroffenen mobilen Knoten auf diese übernommen.

Algorithm 2: *Kopie-Update-Strategie*

ENSURE: // *INPUT:* $ID = \{id_1, \cdots, id_n\}$ *set of client IDs*
ENSURE: // *INPUT:* $RK = \{\{rk_1^1, \cdots, rk_{m_1}^1\}, \cdots, \{rk_n^1, \cdots, rk_{m_n}^n\}\}$
// *set of sets of replication criterions*
ENSURE: // *INPUT: defined fragments*
ENSURE: // *INPUT: UPDATE*

ENSURE: *INVOLVED_FAGMENTS=*$\emptyset$

01. *// Outer Loop: Which fragments are involved in UPDATE*
02. **for each** fragment $\mathcal{F}_i = (f_i, V_i)$ **do**
03.   **if** $f_i \neq UPDATE \circ f_i$ **then**
04.   *// Inner Loop: Which clients use a replicat of this fragment*
05.   **for each** client ID id_j **do**
06.   *// Search stored replication criterions of the client*
07.   **for each** $rk_k^j = (v_{name,k}^j, \Delta_k^{j,max}, \Delta_k^{j,min})$ **do**
08.   **if** $\exists (v_{name}^i \in V_i | \Delta_k^{j,min} \leq \xi_{name}(v_{name,k}^j, v_{name}^i) \leq \Delta_k^{j,max})$ **then**
09.   *// connect client id_j and apply try update*
10.   **if** client with id_j is online **then**
11.   get current rk_{name}^{curr} from client
12.   **if** $rk_{name}^{curr} == v_{name,k}^j$ **then**
13.   apply $UPDATE$ on client
14.   **else**
15.   call algorithm 1 with rk_{name}^{curr} and replicate $RETURN$
16.   **done**
17.   **done**
18. **done**

Da die Funktion dieses Algorithmuses nicht so intuitiv, wie die von Algorithmus 1 ist, wird sie an dieser Stelle ausführlich erläutert.

In der äußeren Schleife (Zeile 02) werden alle Fragmente durchlaufen und festgestellt, welche Fragmente von dem Update betroffen sind (Zeile 03). Ist ein solches Fragment gefunden, werden in einer Unterschleife (Zeile 05) alle registrierten Clients daraufhin untersucht, ob sie ein Replikat des betroffenen Fragmentes besitzen. Dazu werden die gespeicherten Replikationskriteria (Zeile 07), d.h. diejenigen Replikationskriteria, welche der Client beim letzten Replikationsvorgang genutzt hat, dahingehend untersucht, ob sie das betroffene Fragment beschreiben (Zeile 08). Trifft dies für ein Replikationskriterum zu, wird versucht, den entsprechende Client zu kontaktiert (Zeile 10). Ist dieser online, dann wird geprüft, ob das alte, gespeicherte Replikationskriterium noch aktuell ist, oder ob es sich geändert hat (z.B. kann sich der Aufenthaltsort des mobilen Knoten geändert haben.). Liegen keine Änderungen vor, wird das Update übertragen und auf dem mobilen Knoten ausgeführt (Zeile 13). Anderenfalls muss anhand des geänderten Replikationskriteriums erneut repliziert werden (Zeile 15).

Kapitel 5

Implementierung

In diesem Abschnitt wird diskutiert, in welcher Form der hier vorgestellte Replikationsansatz implementiert werden kann.

5.1 Erweiterung von SQL

Um es dem Anwender einfach zu ermöglichen, die in dieser Arbeit vorgestellten und formal spezifizierten Erweiterungen an einer bestehenden Datenbank vorzunehmen, werden im Folgenden die dazu notwendigen SQL-Anweisungen vorgestellt.

ξ-Funktionen, Parameter-Funktionen und Extensionen

In den Ausführungen in Kapitel 3 wird bereits darauf hingewiesen, dass die eigenschaftsbasierte Replikation auf Extensionen und Fragmenten basiert. Da bei der Definition eines Fragmentes diesem bereits vorher definierte Server-Extensionen zugewiesen werden müssen, sind diese Server-Extensionen vor den Fragmenten festzulegen. Vor der Definition einer Server-Extensionen muss eine entsprechende ξ-Funktion implementiert werden. Hierbei ist die Verwendung der `CREATE FUNCTION`-Anweisung [KK01] aus dem SQL99-Standard [MSG01] anwendbar (vgl. Abbildung 5.1). Auf die einzelnen Parameter dieser Standard-Anweisung wird an dieser Stelle nicht näher eingegangen. Der interessierte Leser kann hierzu in [KK01] nachschlagen.

Der Hauptunterschied zwischen Server- und Client-Extensionen war die Möglichkeit, bei Server-Extensionen anstelle diskreter Werte auch Funktionen zur Berechnung von diskreten Werten als Parameter zu verwenden. Diese Parameter-Funktionen werden ebenfalls durch `CREATE FUNCTION`-Anweisungen definiert.

Aufgrund der Ähnlichkeit von Server- und Client-Extensionen werden zuerst untypisierte Extensionen (siehe Abbildung 5.2) erzeugt, welche anschließend, bei der

```
CREATE FUNCTION function_name
[(parameter datatype attributes [,...n])]
RETURNS datatype

  [LANGUAGE {ADA | C | FORTRAN | MUMPS | PASCAL | PLI | SQL}]
  [PARAMETER STYLE {SQL | GENERAL}]
  [SPECIFIC specific_name]
  [DETERMINISTIC | NOT DETERMINISTIC]
  [NO SQL | CONTAINS SQL | READS SQL DATA | MODIFIES SQL DATA]
  [RETURNS NULL ON NULL INPUT | CALL ON NULL INPUT]
  [STATIC DESPATCH]

code block
```

Abbildung 5.1: SQL-Anweisung zum Erzeugen einer ξ-Funktion

Erzeugung eines Fragmentes (siehe Abbildung 5.3) bzw. bei der Replikation, entsprechend spezialisiert werden können. Die SQL-Anweisung zur Erzeugung von Extensionen ist an die `CREATE TABLE` Anweisung angelehnt. Daher ist in diesem Zusammenhang die Verwendung aller, vom DBMS bereitgestellten, Datentypen als Extensions-Parameter zulässig. Offensichtlich ist hierbei `function_name` der Bezeichner der entsprechenden

```
CREATE EXTENSION extension_name
(
  parameter_name datatype[(length)],
  ...n
)
WITH FUNCTION function_name;
```

Abbildung 5.2: SQL-Anweisung zum Erzeugen einer Extension

ξ-Funktion.

Fragmentierungsfunktionen und Fragmente

Nach der Erzeugung der Server-Extensionen muss die Datenbank in Fragmente unterteilt werden. Die dazu notwendige Fragmentierungsfunktion wird direkt in der `CREATE FRAGMENT`-Anweisung (siehe Abbildung 5.3) angegeben und entspricht einer in SQL99 standardisierten `SELECT`-Anweisung. Sie kann beispielsweise durch vorherige Sichtdefinition unterstützt werden.

```
CREATE FRAGMENT fragment_name
AS
  SELECT [ALL | DISTINCT] select_list
  FROM table_name1 [,..., table_nameN]
  [JOIN join_condition]
  [WHERE search_condition]
  [GROUP By group_by_expression]
  [HAVING search_condition]
  [ORDER BY order_expression [ASC | DESC] ]
USING
  extension_name([parameter_wert | parameter_function_name],
                 ...n),
  ...m;
```

Abbildung 5.3: SQL-Anweisung zum Erzeugen eines Fragmentes

Literaturverzeichnis

[Cha96] D. Chamberlin. *Using the New DB2 — IBM's Object-Relational Database System*. Morgan Kaufmann Publishers, San Francisco, CA, 1996.

[CL90] D. Cheu and B. Linden. *SQL-Referenzhandbuch, Version 6.0*. ORACLE Corporation, 500 Oracle Parkway, Redwood City, CA 94065, 1990.

[Dad96] P. Dadam. *Verteilte Datenbanken und Client/Server-Systeme — Grundlagen, Konzepte, Realisierungsstrukturen*. Springer-Verlag, Berlin, 1996.

[Dye99] C. Dye. *Oracle Distributed Systems*. O'Reilly & Associates, Inc., Sebastopol, 1999.

[GMK+98] B. A. Grin, J. G. Milow, A. G. Koslow, L. I. Gusew, M. Lebedew, N. Tschienok, W. Bartenew, W. Kosenko, W. Kasanzewb, N. Iwanow, W. Salizew, and W. Tjobalin. GLONASS Interface Control Document (4-th revision). Technical report, Russian Federation Institute of Space Device Engineering, 1998. http://www.rssi.ru/SFCSIC/public.html.

[Goo92] S. Goodman. *DB-Library Reference Manual*. Sybase Inc., 1992.

[HS00] A. Heuer and G. Saake. *Datenbanken – Konzepte und Sprachen*. MITP, Bonn, 2000.

[Hur93] J. Hurn. *Differential GPS Explained*. Trimble Navigation Ltd., Sunnyvale, California, 1993.

[IN99] T. Imielinski and J. C. Navas. GPS-based geographic addressing, routing, and resource discovery. *Communications of the ACM*, 42(4):86–92, April 1999.

[Kap96] E. D. Kaplan. *Understanding GPS : Principles and Applications*. Artech House Telecommunications Library. Artech House, March 1996.

[KK01] K. Kline and D. Kline. *SQL in a Nutshell – A Desktop Quick Reference*. O'Reilly & Associates, Inc., Sebastopol, 2001.

[LLS00] K. C. K. Lee, H. V. Leong, and A. Si. Incremental View Maintance for Mobile Databases. *Knowledge and Information Systems*, 2:413–437, 2000.

[MSG01] J. Melton, A. R. Simon, and J. Gray. *SQL: 1999 – Understanding Relational Language Components*. Morgan Kaufmann Publishers, San Francisco, California, May 2001.

[Pet97] D. Petković. *Informix Universal Server*. Addison-Wesley, Bonn, 1997.

[PS98] E. Pitoura and George Samaras. *Data Management for Mobile Computing*. Kluwer Academic Publishers, Boston/Dordrecht/London, 1998.

[ZHK96] G. Zhou, R. Hull, and R. King. Generating Data Integration Mediators that Use Materialization. *Journal of Intelligent Information Systems (JIIS)*, 6(2/3):413–437, May 1996.