Replikation und Synchronisation in Oracle Database Lite

Durch die fortdauernde Verbesserung der Rechner- und Speicherleistung mobiler Geräte steigen die Ansprüche an die zu verarbeitenden Informationen. Auch mit mobilen Geräten sollte der Zugriff auf Unternehmensdaten jederzeit, überall, effizient und kostengünstig möglich sein. In diesem Umfeld kommt dem Replikations- und Synchronisationsmodell eine grosse Bedeutung zu. In dieser Arbeit wird das Oracle Database Lite Produkt bezüglich seinem Replikations- und Synchronisationsverfahren beschrieben. In Testszenarien werden Synchronisationskonflikte provoziert und das Verhalten aus Sicht der Mobile Clients beschrieben.

Matthias Hausherr, Titus Jakob, Olivier Rode, Bernhard Wyss | bernhard.wyss@fhnw.ch

Durch die fortdauernde Verbesserung der Rechner- und Speicherleistung mobiler Geräte steigen die Ansprüche an die zu verarbeitenden Informationen. Anwendungen, die bisher auf Desktop-Geräten mit ständiger Verbindung zu den Datenbank-Servern genutzt wurden, werden vermehrt auf mobilen Geräten eingesetzt. Der Zugriff auf Unternehmensdaten sollte, ungeachtet der Natur mobiler Geräte wie eingeschränkte Ressourcen, häufiger und längerer Disconnected Modus, Kommunikationskosten usw. jederzeit, überall, effizient und kostengünstig möglich sein. Durch Replikation wohldefinierter Ausschnitte des zentralen Datenbestandes auf die mobilen Geräte kann dieser Forderung gerecht werden. Trotzdem sollen sich mobile Anwendungen so verhalten, als ob stets auf dem zentralen Datenbestand operiert wird. Im mobilen Umfeld kommt deshalb dem Replikations- und Synchronisationsmodell eine grosse Bedeutung zu [DH00]. Dabei kann sich der Entwickler dem Zielkonflikt der Replikation nicht entziehen [MS04]: Die Forderungen nach Verfügbarkeit, Konsistenz der Daten und Performanz sind nicht gleichzeitig zu erreichen. Wo in diesem Spannungsfeld der drei Forderungen für die jeweilige mobile Anwendung das Optimum gelegt wird, gehört zum sorgfältigen Replikationsdesign und ist abhängig vom konkreten Replikations- und Synchronisationsverfahren des eingesetzten Systems.

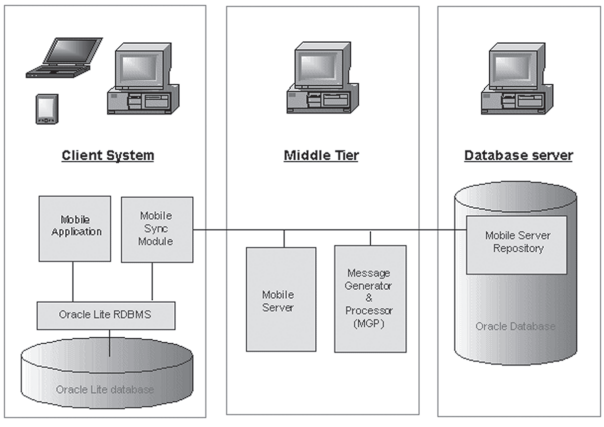

Abb. 1: Oracle Database Lite Architektur [Ora05a]

Für eine hohe Datenverfügbarkeit müssen viele Replikate erstellt werden, die im Disconnected Modus der mobilen Geräte unabhängig voneinander manipuliert werden können. Beim späteren Synchronisationsprozess können so Konflikte auftreten. In dieser Arbeit wird das Verhalten von Oracle Database Lite in Bezug auf die Konfliktbehandlung untersucht.

Im Folgenden wird zuerst die Architektur von Oracle Database Lite vorgestellt. Anschliessend wird auf das Replikations- und Synchronisationsverfahren eingegangen. Nach einer Übersicht über die Testkonfiguration wird anhand verschiedener Testszenarien der Synchronisationsprozess beleuchtet. Zum Schluss werden die Resultate zusammengefasst.

Oracle Database Lite Architektur

Das Oracle Database Lite Produkt ermöglicht die Entwicklung, Verteilung und Verwaltung mobiler Datenbank-Anwendungen. Für die verschiedenen Arbeiten wird eine Reihe von Werkzeugen zur Verfügung gestellt wie das Mobile Development Kit, die Mobile Database Workbench, der Application Packaging Wizard, der Mobile Workspace. Die für die Testumgebung relevanten Werkzeuge werden später dargestellt, einen Überblick gibt [Ora06].

Das System ist als dreischichtige Architektur gestaltet, bestehend aus dem Mobile Client mit dem Oracle Lite Datenbanksystem, dem Mobile Sync Modul und der mobilen Anwendung, dem

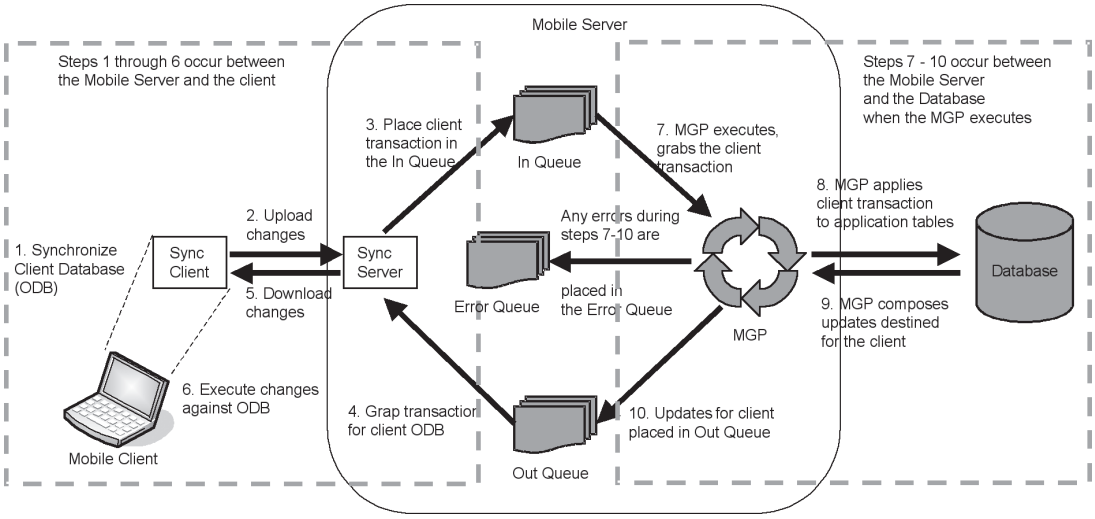

Abb. 2: Synchronisationsprozess [Ora05a]

1. Der Benutzer initiiert eine Synchronisation auf dem mobilen Gerät.
2. Das Mobile Sync Modul fasst alle Änderungen des Clients zu einer Transaktion zusammen und übergibt die Transaktion dem Sync Server im Mobile Server.
3. Der Sync Server platziert die Transaktion in die In-Queue.
4. Der Sync Server holt sich alle Transaktionen, die für den Mobile Client bestimmt sind, aus der Out-Queue.
5. Der Sync Server übergibt diese Transaktionen dem Mobile Sync Module.
6. Das Mobile Sync Module führt alle Änderungen für den Client aus.
7. Der MGP holt sich alle Transaktionen der Mobile Clients aus der In-Queue.
8. Der MGP wendet alle Transaktionen der Mobile Clients gegenüber den entsprechenden Tabellen an.
9. Änderungen, die für die Mobile Clients bestimmt sind, werden durch den MGP je Client zu einer Transaktion zusammengefasst.
10. Der MGP platziert diese Transaktionen in die Out-Queue.

Middle Tier mit dem Mobile Server und dem Message Generator & Processor (MGP) und einem Datenbankserver als Backend mit dem Mobile Server Repository (siehe Abb. 1 aus [Ora05a]).

Eine zentrale Rolle für die Synchronisation spielen der Mobile Server als Sync Server gegenüber den Mobile Clients und der MGP gegenüber dem Backend Datenbanksystem. Der Sync Server reagiert auf Synchronisationsanfragen der Client, der MGP läuft als Hintergrundprozess unabhängig vom Ersteren.

Replikations- und Synchronisationsmodell

Oracle Lite realisiert ein Publish/Subscribe Replikationsmodell mit Hilfe von Snapshots. Für die Anwendungen werden auf dem Mobile Server Publikationen erstellt, die ein oder mehrere Publikationsartikel enthalten können. Diese basieren auf Tabellen oder Views des Datenbankservers. Den Mobile Clients werden diese Publikationen anhand von Subskriptionen zugeordnet. Die Definition der Publikationsartikel erfolgt mittels SQL-Abfragen, die parametrisiert sein können, um benutzerspezifische Ausschnitte (Subsetting) zu definieren.

Mithilfe des Mobile Sync Modules können die mobilen Geräte ihre Snapshots mit den Daten auf dem zugehörigen Datenbankserver synchronisieren. Die Synchronisation zwischen den Oracle Lite Clients und dem Oracle Datenbank Server erfolgt asynchron über den Mobile Server (siehe Abb. 2 aus [Ora05a]). Das Mobile Sync Module arbeitet unabhängig vom MGP und umgekehrt.

Gemäss der Kategorisierung von Replikationsmodellen [GHN96] handelt es sich hier um eine „Lazy Group Replication“, auch bekannt unter der Bezeichnung „Lazy Update Everywhere“ [WPS00]. Änderungen können auf jedem Replikat vorgenommen werden (Group bzw. Update Everywhere), der Abgleich mit den anderen Kopien erfolgt asynchron zu einem beliebigen, späteren Zeitpunkt (Lazy).

Das Wesentliche bei diesem asynchronen Verfahren ist, dass nach den Schritten eins bis sechs im Normalfall dem Mobile Client eine erfolgreiche Synchronisation gemeldet wird, obwohl mögliche Konflikte erst später durch den MGP erkannt und entsprechend den Konfliktregeln aufgelöst werden. Als Konfliktregel können in Oracle Lite Database den Publikationsartikeln entweder „Server Wins“ oder „Client Wins“ hinterlegt werden. In [Ora05b] sind mögliche Synchronisationskonflikte aufgelistet, von denen hier die folgenden drei ausgewählt worden sind, da sie die Grundlage unserer Testszenarien bilden:

1. Der Client und der Server ändern dieselbe Zeile. Dieser Konflikt wird entsprechend den Konfliktregeln durch den Mobile Server aufgelöst.
2. Der Client löscht dieselbe Zeile, die der Server ändert. Dieser Konflikt wird entsprechend den Konfliktregeln durch den Mobile Server aufgelöst.

3. Der Server löscht dieselbe Zeile, die der Client ändert. Dieser Konflikt wird nicht vom Mobile Server aufgelöst. Ein entsprechender Fehler wird generiert und der Administrator muss entscheiden, wie dieser Konflikt aufgelöst werden soll.

Je nach Konfliktregel werden dann bei einer erneuten Synchronisation vorher erfolgreich durchgeführte Änderungen auf dem mobilen Gerät oder dem Server teilweise rückgängig gemacht. Dies geschieht ungeachtet von Transaktions-, ja sogar SQL-Statement-Grenzen. Der Datenabgleich wird für jede geänderte Zeile einzeln durchgeführt. In [PB99] wird dies als Data-Centric-Ansatz im Gegensatz zu einem Transaction-Centric-Ansatz bezeichnet

Testumgebung
Für die nachfolgenden Testszenarien wurde folgende Umgebung eingerichtet:

Computer	Betriebssystem	Installation
Hades	SuSE Linux Enterprise Server 9, Version 9, Patchlevel 3	Oracle Database 10g Oracle Lite Mobile Server
Laptop 1	Windows XP SP 2	Oracle Lite Mobile Development Kit
PC 1	Windows XP SP 2	Oracle Lite Mobile Client Alf
Laptop 2	Windows XP SP 2	Oracle Lite Mobile Client Beat

Die Grundlage des Publikationsartikels und der Subskriptionen ist die Tabelle example, die wie folgt erstellt wurde:

```
CREATE TABLE example(
  Id number(4) primary key,
  Name varchar(30) not null,
  Phone varchar(12));
```

Mit dem Packaging Wizard werden die Publikationsartikel, die gleichzeitig (nicht parametrisierte) Snapshots für die Clients sind, festgelegt:

Reiter	Feld	Input
Plattformen	Ausgewählte Plattform	Oracle Lite WIN32;US
Anwendung	Anwendungsname	Mobile Field Services
	Virtueller Pfad	/MFS
	Beschreibung	Field Task Assignment
	Lokales Anwendungsverz.	<local folder>/mfsdev
Datenbank	Datenbankname	mfs
Snapshots – Neu – Importieren	Tabelle	Example
Snapshots – Neu – Server	Snapshot-Name	Example
	Gewichtung	1
	Eigentümer	DEMO
Snapshots – Neu – Oracle Lite WIN32;US	Auf Client erstellen	Check
	Aktualisierbar?	Check
	Basisobjekttyp	Tabelle
	Konfliktauflösung	Server vorrangig
	Aktualisierungstyp	Schnelle Aktualisierung
	Vorlage	SELECT * FROM DEMO.EXAMPLE
	(Restliche Eingabefelder)	(leer lassen)

Der Zustand auf der Backend-Datenbank wird vor jedem Testszenario mit den folgenden Einträgen neu hergestellt und die beiden Clients damit synchronisiert:

```
INSERT INTO example VALUES(1,
      ,Andre‘, ,111111111111‘);
INSERT INTO example VALUES(2,
      ,Bruno‘, ,222222222222‘);
INSERT INTO example VALUES(3,
      ,Carlo‘, ,333333333333‘);
```

Test Szenarien
Als Grundlage der folgenden Testszenarien betrachten wir ein typisches Anwendungsgebiet: Zur Unterstützung der täglichen Arbeit der Angestellten einer Handelsfirma dient eine zentrale Datenbank. Alle Informationen über Kunden, Anbieter, Bestellungen, Mitarbeiter usw. werden auf einem zentralen Server gehalten. Die Aussendienstangestellten sind mit Laptops ausgestattet. Sie fahren zu den Kunden und nehmen vor Ort deren Bestellungen auf, erfassen oder mutieren Adressen usw., dabei sind sie in der Regel offline und manipulieren einen lokalen Datenbestand. Nach Rückkehr in die Firma werden die mobil erfassten Daten auf den Server übertragen.

In den folgenden Szenarien werden zwei Clients, Alf und Beat, verwendet. Beide Clients führen unabhängig voneinander Aktionen auf ihren lokalen Datenbanken durch und synchronisieren danach in einer bestimmten Reihenfolge. Bei diesen Tests wird kein Data-Subsetting verwendet, weil sich die einzelnen Clients ansonsten gar nie in die Quere kommen können, da sie von vornherein ihre eigenen, nur für sie benutzbaren Bereiche der Tabellen nutzen. Ohne Data-Subsetting können alle Clients auf den gesamten Inhalt der Basistabelle zugreifen, was zwangsläufig zu Konflikten führt.

Es werden keine Manipulationen direkt auf der Backend Datenbank ausgeführt. Die Änderungen des Clients, der zuerst synchronisiert, werden so auf dem Server angewendet, bei der späteren Synchronisation des anderen Clients können Konflikte auftreten.

Bei der Synchronisation werden alle bei einem Client auf der lokalen Datenbank durchgeführten Transaktionen bzw. Änderungen in die In-Queue des Mobile Servers gelegt, wo sie dann auf der Backend Datenbank ausgeführt werden. Die In-

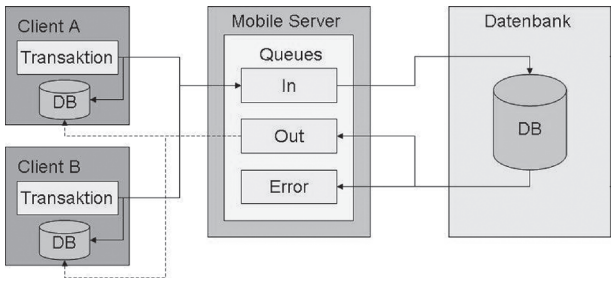

Abb. 3: Situation der Testszenarien

formationen, wo was geändert wurde, wird in die Out-Queue gelegt und bei der nächsten Synchronisation vom Client heruntergeladen, damit die lokale Datenbank mit der Backend Datenbank konsistent bleibt. Falls es auf der Backend Datenbank bei der Ausführung der durch die In-Queue erhaltenen Transaktionen zu einem Konflikt kommt, werden die inkonsistenten Aktionen nicht durchgeführt, sondern werden in die Error-Queue gelegt, wo sie durch einen Administrator bearbeitet bzw. gelöst werden sollten (siehe Abb. 3).

In den Testfällen werden die Updates U1.1, U2, U1.2, ein Insert I4 und ein Delete X1 verwendet. Die dafür verwendeten Statements sehen wie folgt aus:

```
U1.1:  UPDATE example SET
       Phone='updated1' WHERE Id=1;
U1.2:  UPDATE example SET
       Phone='updated3' WHERE Id=1;
U2:    UPDATE example SET
       Phone='updated2' WHERE Id=2;
I4:    INSERT INTO example
       VALUES(4, ,Isert',
       ,444444444444');
X1:    DELETE FROM
       example WHERE Id=1;
```

Testszenario 1

Ausgangslage: Alf macht einen Update U1.1 auf seiner lokalen Tabelle „example" in seiner lokalen Datenbank. Beat ändert denselben Record mit Update U1.2 auf seiner eigenen lokalen Tabelle. Zusätzlich macht er einen Update U2 auf einem anderen Record. Nun synchronisiert Alf vor Beat mit dem Mobile Server.

Alf synchronisiert erfolgreich und seine lokalen Änderungen werden erfolgreich in die Backend Datenbank übernommen und danach für die beiden Clients in der Out-Queue bereit gelegt.

Out-Queue

Alf	Update U1.1
Beat	Update U1.1

Error-Queue

Alf	Keine Einträge
Beat	Keine Einträge

Die nachfolgende Synchronisation mit Beat ergibt ebenfalls die Meldung „Synchronisation erfolgreich". Der Update U1.2 durch Beat landet aber in der Error-Queue, da der betroffene Record bereits durch den Update U1.1 von Alf geändert wurde. Der Update U2 von Beat wird in die Backend Datenbank übernommen und auch in die Out-Queue gelegt.

Out-Queue

Alf	Update U1.1 Update U2
Beat	Update U1.1 Update U2

Error-Queue

Alf	Keine Einträge
Beat	Update U1.2

Resultat: Der MGP erkennt einen Konflikt, den er auflösen kann (Konflikt 1, siehe Replikations- und Synchronisationsmodell), und markiert ihn als „CONFLICT DEDECTED". Nur die in direktem Konflikt stehenden Aktionen gelangen in die Error-Queue. Obwohl der Client Beat die beiden Updates als eine Transaktion ausgeführt hat, gilt die „Alles oder Nichts"-Regel bei der Synchronisation nicht mehr, die Wirkung von U2 bleibt erhalten, die Wirkung von U1.2 wird zurückgesetzt.

Testszenario 2

Ausgangslage: Alf macht einen Update U1.1 auf seiner lokalen Tabelle „example" in seiner lokalen Datenbank. Beat löscht denselben Record mit dem Delete X1 auf seiner eigenen lokalen Tabelle. Zusätzlich macht er einen Update U2 auf einem anderen Record und erstellt einen neuen Record mit dem Statement I4 in derselben Tabelle. Nun synchronisiert Alf vor Beat mit dem Mobile Server.

Alf synchronisiert erfolgreich und seine lokalen Änderungen werden erfolgreich in die Backend Datenbank übernommen und danach für die beiden Clients in der Out-Queue bereit gelegt.

Out-Queue

Alf	Update U1.1
Beat	Update U1.1

Error-Queue

Alf	Keine Einträge
Beat	Keine Einträge

Die nachfolgende Synchronisation mit Beat ergibt auch die Meldung „Synchronisation erfolgreich“. Der Delete X1 durch Beat landet aber in der Error-Queue, da der betroffene Record bereits durch einen Update U1.1 von Alf geändert wurde. Die weiteren Mutationen U2 und I4 von Beat werden in die Backend Datenbank übernommen und auch in die Out-Queue gelegt.

Out-Queue

Alf	Update U1.1 Update U2 Insert I4
Beat	Update U1.1 Update U2 Insert I4

Error-Queue

Alf	Keine Einträge
Beat	Delete X1

Resultat: Der MGP erkennt einen Konflikt, den er auflösen kann (Konflikt 2, siehe Replikations- und Synchronisationsmodell), und markiert ihn als „CONFLICT DEDECTED“. Nur die in direktem Konflikt stehenden Aktionen gelangen in die Error-Queue. Obwohl der Client Beat sein Delete, Update und Insert als eine Transaktion ausgeführt hat, gilt die „Alles oder Nichts“-Regel bei der Synchronisation nicht mehr, die Wirkung von U2 und I4 bleibt erhalten, die Wirkung von X1 wird zurückgesetzt.

Testszenario 3

Ausgangslage: Alf löscht einen Record mit dem Statement X1 auf seiner lokalen Tabelle „example“. Gleichzeitig ändert Beat auf seiner lokalen Tabelle „example“ denselben Record mit einem Update U1.1, fügt einen weiteren Record mit Insert I4 der Tabelle hinzu und macht einen weiteren Update U2 auf einer weiteren Zeile in der Tabelle. Nun synchronisiert Alf vor Beat.

Nach der Synchronisation von Alf mit dem Mobile Server erscheint die Meldung „Synchronisation erfolgreich“ und die durch Alf vollzogene Änderung wird zuerst in die In-Queue des Mobile Servers übernommen und dann auf die Backend Datenbank appliziert. Danach wird die Änderung für die einzelnen Clients in die Out-Queue des Mobile Servers gelegt. Fehler sind dabei keine aufgetreten.

Out-Queue

Alf	Keine Einträge
Beat	Delete X1

Error-Queue

Alf	Keine Einträge
Beat	Keine Einträge

Bei der Synchronisation mit Beat erscheint wieder eine „Synchronisation erfolgreich“ Meldung, obwohl der Update U1.1 in der Error-Queue landet, da der zu mutierende Record bereits durch den Delete X1 von Alf in der Hauptdatenbank nicht mehr existiert. Auffällig dabei ist, dass alle weiteren Aktionen von Beat in der Error-Queue landen, auch wenn sie in keinem direkten Konflikt mit den Änderungen von Alf stehen. Wenn Beat nun ein zweites Mal synchronisiert, wird seine lokale Datenbank mit der Backend Datenbank überschrieben. Die Folge ist, dass seine lokalen Änderungen verloren gehen. D. h. der Delete X1 wird auf seiner lokalen Datenbank ausgeführt und seine Änderungen U1.1 und I4 werden durch die Backend Datenbank überschrieben und gehen somit verloren.

Out-Queue

Alf	Keine Einträge
Beat	Keine Einträge

Error-Queue

Alf	Keine Einträge
Beat	Update U1.1 Update U2 Insert I4

Resultat: Der MGP erkennt einen Konflikt, den er nicht auflösen kann (Konflikt 3, siehe Replikations- und Synchronisationsmodell), und markiert ihn als „ERROR“. Alle, also auch Aktionen, die nicht direkt in Konflikt stehen, werden nicht ausgeführt, sondern landen in der Error-Queue.

Zusammenfassung

Mit einfachen Testszenarien haben wir das Verhalten von Oracle Database Lite beim Auftreten von drei Synchronisationskonflikten dargestellt. Das Verhalten wurde aus Sicht zweier Mobile Clients beschrieben, obwohl die Konflikte immer nur zwischen einem Client und dem Server auftreten. In dieser künstlich erzeugten symmetrischen Konstellation zeigt sich am stärksten die asymmetrische Natur der Konfliktauflösung. Der UP-

DATE-UPDATE-Konflikt (Testszenario1) ist eher symmetrisch, die Reihenfolge der Synchronisation spielt keine Rolle. Der Konflikt wird mit der definierten Regel (hier Server Wins) aufgelöst. Der DELETE-UPDATE-Konflikt (Testszenarien 2 und 3) ist aus Sicht der Clients asymmetrisch, je nach Reihenfolge der Synchronisation wird der Konflikt aufgelöst oder der Datenabgleich führt zu einem Fehler. Wenn Konflikte erkannt und aufgelöst werden (Testszenario 1 und 2), ist es möglich, dass die Transaktionsgrenzen der Mobile Clients verschwinden, die „Alles oder Nichts"-Regel gilt nicht mehr, das Verfahren ist Data-Centric. Durch das asynchrone Verfahren erkennen die Mobile Clients nicht, ob ihr Datenabgleich zu Konflikten geführt hat.

Dies mag aus Sicht der Mobile Clients eher nachteilig erscheinen. Man muss sich aber bewusst sein, dass ein auf Performanz, insbesondere Skalierbarkeit ausgelegtes Replikations- und Synchronisationsverfahren, wie es das Oracle Database Lite bietet, bei der Forderung nach Konsistenz bzw. der Forderung nach Datenverfügbarkeit gewisse Abstriche machen muss (Zielkonflikt). Es liegt vollständig in der Hand des Entwicklers mobiler Anwendungen, welchen der beiden Forderungen mehr Priorität eingeräumt werden soll.

Zielt die Anwendung auf möglichst hohe Datenverfügbarkeit und wird deshalb auf Subsetting verzichtet, kann es zu Situationen kommen, wie sie in den Testszenarien beschrieben sind. Die Datenkonsistenz ist dabei insofern garantiert, als dass jeder Mobile Client für sich ein konsistentes Abbild der Daten hält, das aber verschieden sein kann von anderen Clients oder des Servers.

Vorteilhafter wäre es, mit einem sorgfältigen Replikationsdesign für die mobile Anwendung von vornherein Synchronisationskonflikte zu vermeiden. Es wird kaum nötig sein, allen Mobile Clients die vollständigen Publikationsartikel zur Änderung zur Verfügung zu stellen. Mithilfe parametrisierter Snapshots können den Mobile Clients nur die für sie relevanten Ausschnitte der Publikationsartikel bereitgestellt werden (Subsetting). Je kleiner die Menge der gemeinsamen änderbaren Zeilen ist, je geringer ist die Wahrscheinlichkeit eines Synchronisationskonflikts.

Referenzen

[DH00] Dunham, M., Helal, A.: Mobile Computing and Databases: Anything New?. SIGMOD RECORDS, 4(4), pp. 311-321, 2000.

[GHN96] Gray, J., Helland, P., O'Neil, P., Shasha, D.: The Dangers of Replication and a Solution, Proceedings of the 1996 ACM SIGMOD International Conference on Management of Data, Montreal, Quebec, Canada, pp. 173 – 182, ACM, 1996.

[MS04] Mutschler, B., Specht, G.: Mobile Datenbanksysteme, Springer, Berlin, 2004.

[Ora05a] Oracle® Database Lite Developer's Guide 10g (10.2.0) Part No. B15920-01, 2005.

[Ora05b] Oracle® Database Lite Administration and Deployment Guide 10g (10.2.0) Part No. B15921-01, 2005.

[Ora06] Oracle Database Lite 10gR2 Technical White Paper, 2006

[PB99] Phatak, S. H., Badrinath, B. R.: Conflict resolution and reconciliation in disconnected databases. Proceedings of MDDS 1999, Sep. 1999

[WPS00] Wiesmann, M., Pedone, F., Schiper, A., Kemme, B., Alonso, G.: Understanding Replication in Databases and Distributed Systems, Proceedings of 20th International Conference on Distributed Computing Systems, 2000.