Durch die Integration neuer Produkte in das Oracle-Portfolio gibt es immer einige, denen strategisch eine neue, wichtige Rolle zufällt, und andere, die vor dem produkt-technischen Renten-Eintritt in Bezug auf ihren Lebenszyklus stehen. Eine solche Situation hat sich mit dem Zukauf der Berkeley DB (ehemals „Sleepycat") für die Oracle-Lite-Datenbank ergeben.

Was kommt, wenn Oracle Lite geht?

Karin Patenge, ORACLE Deutschland B.V. & Co. KG

Im Juni 2010 wurden die Komponenten „Oracle Lite Client Database", „Branch Office" und „Web-to-Go", die zum Oracle Lite Mobile Server gehörten, in den Maintenance-Modus versetzt. Seit November 2010 ist „Oracle Lite Client Database" zudem nicht mehr auf der Oracle-Preisliste. Dieser Artikel zeigt auf, welche Alternativen für die lokale Speicherung von Daten auf mobilen Clients nunmehr zur Verfügung stehen und wie die Synchronisation mit der Oracle-Datenbank im Back-End zukünftig weiterhin möglich ist.

Oracle Lite ist vor allem als eine schlanke, SQL92-konforme, relationale Datenbank-Engine bekannt, die auf unterschiedlichen Geräten (wie Laptop, PDA etc.) und verschiedenen Plattformen (Microsoft Windows, Linux und Symbian) zum Einsatz kam. Ein mögliches Einsatz-Szenario war dabei die direkte Einbettung von Oracle Lite in die Anwendung. Bei deren Start kamen die Oracle-Lite-Datenbank-Libraries zum Anwendungsprozess hinzu.

Entwickelt werden konnte auf Basis vorhandener ODBC-, JDBC-, ADO.NET- oder SODA-Schnittstellen in verschiedenen Programmiersprachen wie Java, C/C++ oder Visual Basic. Für die Kommunikation der Anwendung mit dem Back-End, insbesondere um Daten zwischen mobilem Client und dem Back-End auszutauschen, sorgte auf der Clientseite ein Synchronisationsprozess. Der „msync" genannte Prozess lief als eigenständige Anwendung ebenfalls auf dem mobilen Gerät. Sein Gegenstück war ein entsprechender Synchronisationsprozess im Mobile Server. Dieses Konstrukt bewirkte, dass Daten offline auf mobilen Geräten zur Verfügung gestellt, dort geändert und bei vorhandener Netzwerkverbindung wieder mit einer Datenzentrale abgeglichen werden konnten (siehe Abbildung 1). Daran hat sich vom Prinzip her auch mit der Abkündigung des Oracle-Lite-Database-Client nichts geändert. Dieser wurde mit der Berkeley DB ersetzt oder optional auch durch SQLite [2] als Mobile-Client-Database (siehe Abbildung 2).

Berkeley DB – die kleine Unbekannte

Die Berkeley DB gibt es bereits seit Anfang der 1990er Jahre als sogenanntes „Embedded Transactional Data Management System", konzipiert für den Einsatz auf Geräten unterschiedlichster Größenordnung von Mobiltelefonen über WLAN-Router bis zu Servern sowie für unterschiedlichste Einsatzbereiche. Die Berkeley DB ist dabei zum Zwecke der lokalen Datenhaltung integraler Bestandteil der Anwendung. Es gibt keine separat laufenden Prozesse für die Speicherung oder Verarbeitung der Daten beziehungsweise für das Management der Datenbank selbst (siehe Abbildung 3). Dadurch ist sie für die Nutzer einer Anwendung als solche zu-

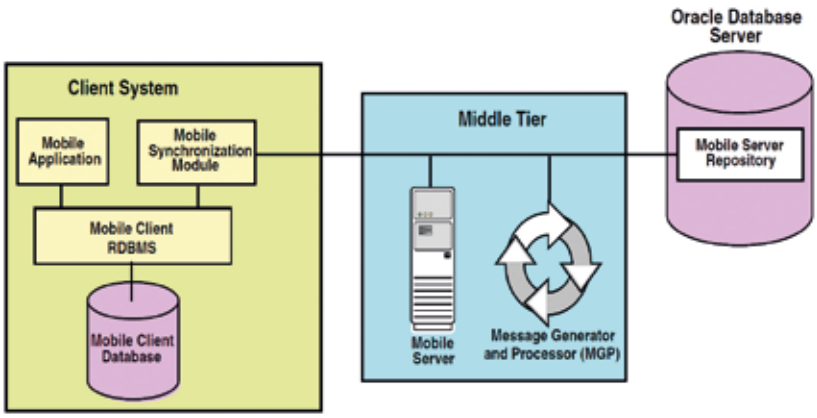

Abbildung 1: Die Architektur von Oracle Database Lite 10g (Ausschnitt aus [1])

Abbildung 2: Mobile Anwendungen mit lokaler Datenhaltung

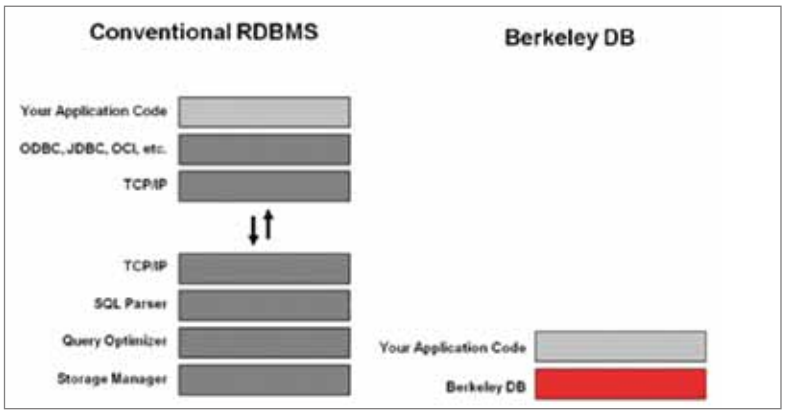

Abbildung 3: Vergleich konventionelles RDBMS mit Embedded DB

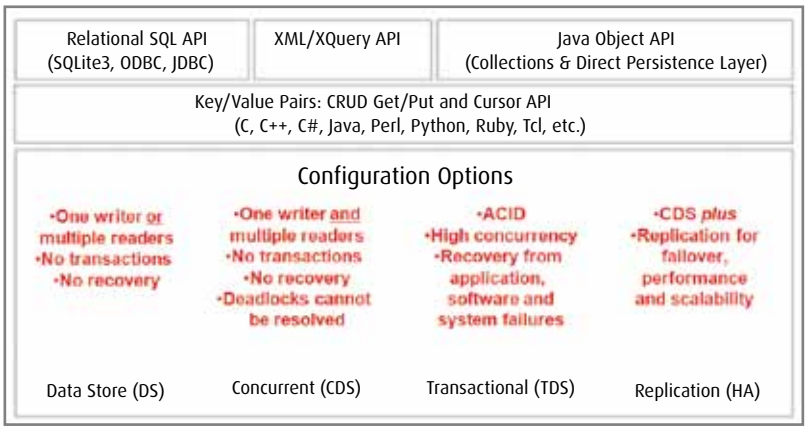

Abbildung 4: Aufbau der Berkeley DB

```
D:\Workdir\BDB>sqlite3 bdb_hr_test.db
Berkeley DB 11g Release 2, library version 11.2.5.3.15: (December 19, 2011)
Enter ".help" for instructions
Enter SQL Statements terminated with a ";"
dbsql> _
```

Abbildung 5: Aufruf des Berkeley DB SQL API auf Kommandozeilen-Ebene mit sqlite3 alias dbsql

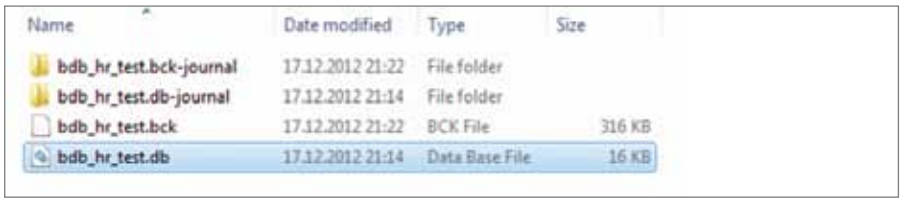

Abbildung 6: Mit sqlite3 erzeugte .db-Datei und zugehöriger „-journal"-Ordner

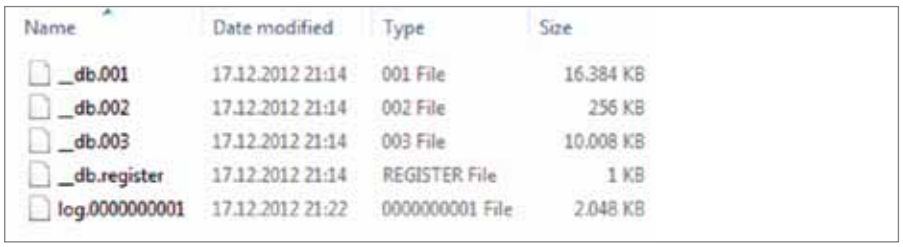

Abbildung 7: Inhalt des „-journal"-Ordners

meist gar nicht erkennbar. Verborgen hinter der Applikation verrichtet sie ihre Arbeit so, wie die Anwendung es von ihr einfordert.

Egal, ob ausschließlich als In-Memory-Datenhaltung oder mit Persistierung, ob ohne oder mit Replikation, ob für Mehrbenutzer-Betrieb geeignet oder nicht, die Nutzung kann flexibel den Anforderungen angepasst werden. So verwundert es nicht, dass Googles Account-Informationen und -Präferenzen in der Berkeley DB gespeichert sind, ebenso Dienste, die auch Amazon in Anspruch nimmt; Mobiletelefone (unter anderem von Motorola) haben sie integriert und sie ist die Storage Engine für MySQL.

Der fundamentale Unterschied zu Oracle Lite besteht darin, dass die Berkeley DB zu den sogenannten „NoSQL-Datenbanken" zählt. NoSQL steht dabei für „Not only SQL" und hat sich als Sammelbegriff für alle nicht-relationalen Datenhaltungssysteme etabliert. Die Berkeley DB in ihrem Kern ist dabei ein Vertreter der sogenannten „Key-Value-Stores". Datensätze werden hier als Schlüssel und zugehöriger Wert abgelegt, die Werte über das Auffinden des entsprechenden Schlüssels wieder ausgegeben. Sowohl Schlüssel als auch Wert können beliebige Informationen (auch Objekte) beinhalten, sind also an keine vordefinierten Datentypen gebunden. Das heißt, die Berkeley DB ist schemalos. Sie unterstützt ACID-Transaktionen für schreibende Zugriffe ebenso wie referenzielle Integrität, In-Memory-Datenhaltung (disk-less), Recovery, Replikation, Multi-Threading und Multi-Processing, Online-Backups, Kompression oder auch Verschlüsselung. Sie wird über Programmier-Schnittstellen angesprochen, die unter anderem für Sprachen wie C, C++, Java, C#, Perl, Python, PHP, Tcl oder Ruby zur Verfügung stehen. Im Plattformbereich werden 32- und 64-bit-Geräte unterstützt sowie verschiedene Betriebssysteme wie Linux, Windows, Android oder iOS.

Damit die Berkeley DB für die Kommunikation mit dem Mobile Server und der relationalen Datenbank im Back-End nutzbar ist, wurde sie um eine relationale Schnittstelle, das SQL-

```
-------------------------------------------------------
-- BDB Erweiterung zu SQLite:
--
-- Sequences  -> “The SQL API sequence support is a partial implementation
--   of the sequence API defined in the SQL 2003 specification.“
--   Alternativ kann AUTOINCREMENT (SQLite) bei Primary Key-Spalten benutzt werden.
-------------------------------------------------------

-------------------------------------------------------
--  Drop tables and sequences
-------------------------------------------------------
DROP TABLE LOCATIONS;
DROP TABLE REGIONS;
SELECT drop_sequence(“SEQ_REGIONS“);
SELECT drop_sequence(“SEQ_LOCATIONS“);
.tables

-------------------------------------------------------
--  DDL to create Tables REGIONS and LOCATIONS
-------------------------------------------------------
CREATE TABLE REGIONS
(      REGION_ID NUMBER,
       REGION_NAME VARCHAR2(25),
       CONSTRAINT REG_ID_PK PRIMARY KEY (REGION_ID)
) ;

-- alternativ:
CREATE TABLE IF NOT EXISTS REGIONS
(      REGION_ID INTEGER PRIMARY KEY AUTOINCREMENT,
       REGION_NAME VARCHAR2(25)
) ;

CREATE TABLE LOCATIONS
(      LOCATION_ID NUMBER(4,0),
       STREET_ADDRESS VARCHAR2(40),
       POSTAL_CODE VARCHAR2(12),
       CITY VARCHAR2(30) NOT NULL,
       STATE_PROVINCE VARCHAR2(25),
       COUNTRY_ID CHAR(2),
       CONSTRAINT LOC_ID_PK PRIMARY KEY (LOCATION_ID),
       CONSTRAINT LOC_C_ID_FK FOREIGN KEY (COUNTRY_ID) REFERENCES COUNTRIES (COUNTRY_ID)
) ;
.tables

-------------------------------------------------------
--  DDL to create Sequences using SQL API feature
-------------------------------------------------------
SELECT create_sequence(“SEQ_LOCATIONS“,“start“,1000,“incr“,100,“maxvalue“,9000);
SELECT create_sequence(“SEQ_REGIONS“,“start“,1,“incr“,1,“maxvalue“,99999999);

-------------------------------------------------------
--  DDL to create Indexes
-------------------------------------------------------
CREATE UNIQUE INDEX LOC_ID_PK ON LOCATIONS (LOCATION_ID) ;
CREATE INDEX LOC_CITY_IX ON LOCATIONS (CITY);
CREATE INDEX LOC_STATE_PROVINCE_IX ON LOCATIONS (STATE_PROVINCE);
CREATE INDEX LOC_COUNTRY_IX ON LOCATIONS (COUNTRY_ID);
CREATE UNIQUE INDEX REG_ID_PK ON REGIONS (REGION_ID);
.indices

-------------------------------------------------------
--  DML for Table REGIONS
--    using sequences to generate primary key values
-------------------------------------------------------
BEGIN TRANSACTION;
INSERT INTO REGIONS (REGION_ID, REGION_NAME) VALUES (nextval(“SEQ_REGIONS“),‘Europe‘);
```

```
INSERT INTO REGIONS (REGION_ID, REGION_NAME) VALUES (nextval("SEQ_REGIONS"),'Americas');
INSERT INTO REGIONS (REGION_ID, REGION_NAME) VALUES (nextval("SEQ_REGIONS"),'Asia');
INSERT INTO REGIONS (REGION_ID, REGION_NAME) VALUES (nextval("SEQ_REGIONS"),'Middle East and Africa');
COMMIT;

------------------------------------------------------------
--  View Table Contents
------------------------------------------------------------
SELECT * FROM LOCATIONS;
SELECT * FROM REGIONS;
```

Listing 1: Auszug aus Beispielskript für SQL-API – Tabellen, Indizes, Sequenzen & CLI

API, erweitert (siehe Abbildung 4). Diese Schnittstelle ist zu fast 100 Prozent SQLite3-konform. Somit ist die Berkeley DB Code-neutral für SQLite3 einsetzbar. Performance-Vorteile [4] bei konkurrierenden Zugriffen, insbesondere durch das Locking-Verhalten (Page Level Locking), die Skalierbarkeit in Bezug auf große Datenmengen und viele gleichzeitige Zugriffe, ebenso wie der Support aus einem Haus sprechen hier für die Berkeley DB.

Beide, sowohl SQLite als auch Berkeley DB, unterstützen in wesentlichen Teilen den SQL92-Standard. Eine Liste der von SQLite3 nicht unterstützten Features im Standard findet sich auf den Webseiten von SQLite.org [3]. Die Unterschiede zwischen dem SQL-API der Berkeley DB auf der einen und SQLite3 auf der anderen Seite sind in [5] beschrieben.

Wer nicht sofort mit einer höheren Programmiersprache gegen die Berkeley DB programmieren möchte, kann zunächst auch das im SQL-API mitgelieferte Kommandozeilenwerkzeug „dbsql" ausprobieren. Dieses entspricht dem von SQLite3 bekannten Werkzeug „sqlite3". Installiert man, wie im Handbuch [6] beschrieben, die Berkeley DB mit dem SQL-API (unter Windows kann „dbsql.exe" in „sqlite3.exe" und „libdb_sql5x.dll" in „sqlite3.lib" umbenannt werden), lassen sich die Möglichkeiten und Einschränkungen recht einfach nachvollziehen (siehe Abbildung 5).

Zur besseren Veranschaulichung dienen die nachfolgenden Skript-Ausschnitte (siehe Listings 1 und 2). Sie wurden einem Skript entnommen, das das Beispielschema HR der Oracle-

```
------------------------------------------------------------
--  Sample DDL to create Trigger UPDATE_JOB_HISTORY
------------------------------------------------------------
CREATE TRIGGER IF NOT EXISTS UPDATE_JOB_HISTORY
  AFTER UPDATE ON EMPLOYEES
  FOR EACH ROW
BEGIN
  INSERT INTO JOB_HISTORY (EMPLOYEE_ID, START_DATE, END_DATE, JOB_ID,
DEPARTMENT_ID)
    VALUES(old.EMPLOYEE_ID, old.HIRE_DATE, date('now'), old.JOB_ID,
old.DEPARTMENT_ID);
END;

------------------------------------------------------------
--  View Table Contents
------------------------------------------------------------
SELECT * FROM LOCATIONS;
SELECT * FROM REGIONS;

------------------------------------------------------------
--  Sample DDL to create Views
------------------------------------------------------------
CREATE VIEW VW_EMPLOYEES AS
SELECT
  e.EMPLOYEE_ID,
  e.FIRST_NAME,
  e.LAST_NAME,
  d.DEPARTMENT_NAME
FROM
  EMPLOYEES e LEFT OUTER JOIN DEPARTMENTS d
  ON (d.DEPARTMENT_ID = e.DEPARTMENT_ID);

SELECT * FROM VW_EMPLOYEES ORDER BY EMPLOYEE_ID;

------------------------------------------------------------
--  Backup & Restore
------------------------------------------------------------
.backup main hr_test.bck
.restore main hr_test.bck

------------------------------------------------------------
--  More CLI commands
------------------------------------------------------------
.help
.databases
.dump regions
```

Listing 2: Auszug aus Beispielskript für SQL-API – Trigger, Views & CLI

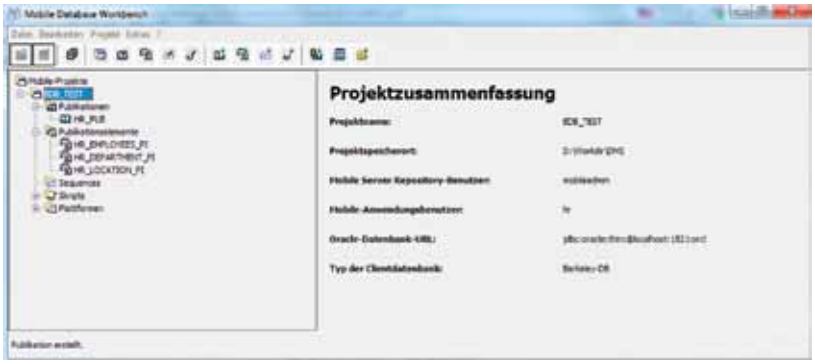

Abbildung 8: Eine Publikation, erstellt mit der Mobile-Server-Komponente „Mobile Database Workbench“

Abbildung 9: Messgerät

Datenbank SQL-API-konform umschreibt. Neben Sequenzen lassen sich auch Views oder Trigger anlegen. Darüber hinaus stehen Befehle für Backup & Restore zur Verfügung (siehe Listing 2).

Im Ergebnis ist eine Datenbank-Datei „bdb_hr_test.db“ entstanden. Automatisch wurde dabei ein Ordner mit dem Namen der .db-Datei und dem Suffix „-journal“ angelegt (siehe Abbildung 6). Dieser Ordner enthält die sogenannten „Environment-Dateien“, die den Zugriff auf die Datenbank (der Berkeley-DB-Begriff für eine Datenbank ist „Environment“) über mehrere Prozesse hinweg ermöglichen. Sie sollten nach Möglichkeit nicht gelöscht werden (siehe Abbildung 7).

Der Mobile Server

Sind zwischen dem Datenbank-Back-End und dem mobilen Endgerät Daten zu synchronisieren, kommt der Mobile Server zum Einsatz. Oracle Lite kann als Client-Datenbank dabei nur noch bis Version 10.3.0.3 angebunden werden. In der aktuellen Version 11.2 werden SQLite und die Berkeley DB unterstützt. Die Daten-Synchronisation erfolgt manuell oder automatisiert und kann dabei sowohl vom Server als auch vom Client initiiert werden. Grundlage dafür ist ein Publish-Subscribe-Modell, nach der Publikationen erstellt werden, die Publikationselemente beinhalten und die von einem Nutzer bezogen werden können (siehe Abbildung 8).

Die Installation des Mobile Servers setzt einen J2EE-konformen Application Server voraus. Zertifiziert werden dabei die jeweils aktuellen Versionen von Oracle WebLogic, Oracle Application Server sowie GlassFish. Die Administration erfolgt über Admin-Konsole, die mittels „http://<host>:<port>/mobile“ aufgerufen wird.

Wie Berkeley DB, mobiles Gerät und Mobile Server zusammenspielen, zeigt ein Video auf YouTube [7] sehr anschaulich am Beispiel eines medizinischen Gerätes, das die Pulsfrequenz aufzeichnet und die Sauerstoffsättigung des Blutes misst (siehe Abbildungen 9 und 10).

Migration in acht Schritten

Wer bestehende mobile Anwendungen von Oracle Lite nach Berkeley DB migrieren möchte, kann sich an das im Kapitel 6 des Mobile-Server-Installationshandbuchs beschriebene Vorgehen halten:

- Backup der Oracle-Lite-Datenbank erstellen

Abbildung 10: Mobile Anwendung mit gemessenen Daten

Abbildung 11: Oracle Mobile Enterprise Application Platform

- Upgrade „Oracle Lite Mobile Client“ auf Version 10.3.0.3.0 (enthält Werkzeug für den Export der Oracle-Lite-Datenbank)
- Export der Oracle-Lite-Datenbank
- Migration der Oracle-Lite-Publikation(en)
- Upgrade des „Oracle Database Lite Mobile Server 10.3.0.3.0“ auf „Oracle Database Mobile Server 11*g*“
- Anlegen von Berkeley-DB-basierten Publikationen und Erstellen der mobilen Anwendung
- Installieren der neuen mobilen Anwendung
- Import der Daten in die Berkeley DB

Mobil ist nicht immer gleich Online

Auch in Zeiten der Maschine-zu-Maschine-Kommunikation sowie der fast flächendeckend verfügbaren Kommunikationsnetze, die einen Online-Zugriff auf die zentrale Datenbasis von Unternehmen ermöglichen, ist dieser Zugriff doch nicht durchgängig zu jeder Zeit, an jedem Ort und unter allen Umständen realisierbar. Dadurch wird weiterhin die Notwendigkeit bestehen, Applikationen und die von diesen benötigten Daten auf mobilen Geräten als Arbeitsgrundlage zur Verfügung zu stellen.

Sind die Arbeiten abgeschlossen und/oder stehen LAN oder WLAN wieder zur Verfügung, werden die Daten in den Unternehmensdatenbestand zurückgeschrieben. Was dafür an technischen Komponenten benötigt wird, stellt Oracle mit der Berkeley DB und dem Mobile Server bereit (siehe Abbildung 11).

Weitere Informationen

- Berkeley DB auf OTN: http://www.oracle.com/technetwork/database/berkeleydb
- Mobile Server auf OTN: http://www.oracle.com/technetwork/products/database-mobile-server/overview/index.html

Verweise

[1] http://docs.oracle.com/cd/E12095_01/doc.10303/e12090.pdf
[2] http://www.sqlite.org
[3] http://www.sqlite.org/omitted.html
[4] http://www.oracle.com/technetwork/database/berkeleydb/learnmore/bdbvssqlite-wp-186779.pdf
[5] http://www.oracle.com/technetwork/database/berkeleydb/bdb-sqlite-comparison-wp-176431.pdf
[6] http://docs.oracle.com/cd/E17076_02/html/bdb-sql/BDB-SQL-Guide.pdf
[7] http://www.youtube.com/watch?gl=GB&v=wp2lPEK27P4

Karin Patenge
karin.patenge@oracle.com