DAO - DBC - eSQL

Insy-Ausarbeitung

Erik Brändli

29. Mai 2016

Contents

1 Einleitung

Diese Ausarbeitung behandelt Datenbankzugriffsmethoden und Datenabbildung.

2 ODBC

Open Database Connectivity ist Microsofts strategische Schnittstelle für den Zugriff auf Daten in einer heterogenen Umgebung aus relationalen und nicht-relationalen Datenbank-Managementsystem. Auf der Grundlage der Call Level Interface-Spezifikation der SQL Access Group bietet ODBC eine offene, anbieterunabhängige Möglichkeit für den Zugriff auf Daten, die in einer Vielzahl von Computer-Datenbanken gespeichert wurden.

ODBC bietet jetzt eine universelle Schnittstelle für den Datenzugriff unabhängig von der gegebenen Datenlandschaft. Mit ODBC, können Entwickler eine Anwendung gleichzeitig zugreifen, anzeigen und ändern von Daten aus mehreren unterschiedlichen Datenbanken.

ODBC ist eine zentrale Komponente der Microsoft Windows Open Services Architecture. Mit wachsenden branchenweite Unterstützung entsteht ODBC schnell als wichtiger Branche für Datenzugriff für Windows und Macintosh.

2.1 ODBC-Client

Der Client muss die ODBC-API verwenden (siehe Verwendung). Die API ist stadardisiert und kann stets über die gleichen Methoden verwendet werden.
Dies funktioniert nur wenn die standardisierten Befehle durch den Treiber geleitet werden. Der Client-Code ansich kann natürlich nicht vom DBMS verstanden werden.

2.2 ODBC-Treiber

Der Treiber ist zuständig für die Umwandlung der Befehle aus dem Client-Code in ein verständliches Format für das DBMS. Danach wird dies übers Netzwerk ans DBMS übermittelt. Die Antwort wird ebenfalls wieder vom Treiber in ein geeignetes Format übersetzt und an die Client-Applikation über API definierte Rückgabewerte weitergegeben.
ODBC Treiber gibt es nahezu für jedes DBMS, jedoch sind nicht alle OpenSource bzw. frei zur Verwendung (Lizenzen).

2.3 Verwendung

Hier ist ein lauffähiges C# Programm, das ODBC verwendet um Abfragen auf die Datenbank zu machen.

```
using ...;
using System.Data.Odbc;

namespace ConsoleApplication1{
    class Program{
        static void Main(string[] args){
            if (args.Length != 1 ){
                Console.WriteLine("usage: program connection-string");
                return;
            }
            String connStr = args[0];
            OdbcConnection DbConnection = new OdbcConnection( connStr );
            try{ DbConnection.Open();
            }catch (OdbcException ex){ Console.WriteLine(ex.Message);
                return;
            }
            String query = Console.ReadLine(); //Input of SQL in Console
            OdbcCommand DbCommand = DbConnection.CreateCommand();
            DbCommand.CommandText = query;
            try{
                OdbcDataReader DbReader = DbCommand.ExecuteReader();
                do{
                    int fCount = DbReader.FieldCount;
                    if (fCount > 0){
                        Console.Write(":");
                        for (int i = 0; i < fCount; i++){
                            String fName = DbReader.GetName(i);
                            Console.Write(fName + ":");
                        }
                        Console.WriteLine();
                        while (DbReader.Read()){
                            Console.Write(":");
                            for (int i = 0; i < fCount; i++){
                                String col = DbReader.GetString(i);
                                Console.Write(col + ":");
                            }
                        }
                    }else{Console.WriteLine("Query affected " + DbReader.RecordsAffected
                         + " row(s)");}
                }
                while (DbReader.NextResult());
                DbReader.Close();
            }catch (OdbcException ex){
                Console.WriteLine(ex.Message);
                return;}
            DbCommand.Dispose();
            DbConnection.Close();}}}
```

3 JDBC

JDBC (Java Database Connectivity) ist ein Java-API zur Ausführung von SQL-Anweisungen. JDBC bietet eine Standard-Schnittstelle für Entwickler von Datenbankwerkzeugen und ermöglicht das Programmieren von Datenbankanwendungen unter Verwendung von reinem Java.

Durch JDBC wurde eine Möglichkeit geschaffen, SQL Anfragen an nahezu jedes beliebige relationale Datenbanksystem zu senden. Dadurch entfällt die Notwendigkeit den Quellcode eines einmal geschriebenen Programms, beim Wechsel auf ein anderes Datenbanksystem (z.B. von Oracle nach MySQL), anpassen zu müssen. Der Datenbankzugriff funktioniert unabhängig von der verwendeten Datenbank immer auf die gleiche Weise. Ein weiterer Vorteil bei der Verwendung von Java und JDBC ist die Plattformunabhängigkeit, ein geschriebenes Programm ist demnach nicht nur unabhängig vom verwendeten Datenbanksystem, sondern zugleich auch noch unabhängig von der zugrundeliegenden Plattform, zumindest dann wenn für das verwendete Betriebssystem eine JVM zur Verfügung steht. Zu den Aufgaben von JDBC gehört es, Datenbankverbindungen aufzubauen und zu verwalten, SQL-Anfragen an die Datenbank weiterzuleiten und die Ergebnisse in eine für Java nutzbare Form umzuwandeln und dem Programm zur Verfügung zu stellen. Um JDBC zu nutzen muss für das verwendete Datenbanksystem ein entsprechender Treiber zur Verfügung stehen, der die JDBC-Spezifikationen implementiert. Diese Treiber stehen für alle gängigen Datenbanksysteme zur Verfügung und werden meist von deren Herstellern direkt mitgeliefert.[1]

3.1 Treibertypen

Ein JDBC-**Typ-1**-Treiber kommuniziert ausschließlich über einen JDBC-ODBC-Bridge-Treiber. Die bekannteste JDBC-ODBC-Bridge ist die von Sun vertriebene. Damit ist ein Typ-1-Treiber abhängig von einem installierten ODBC-Treiber. Der JDBC-ODBC-Bridge-Treiber wandelt JDBC- in ODBC-Anfragen um. Ein Typ-1-Treiber wird dann verwendet, wenn es zu der Datenbank einen ODBC-Treiber, jedoch keine eigenständigen JDBC-Treiber gibt. Mit Java 9 wird die Unterstützung für JDBC-Typ-1-Treiber eingestellt.

Ein **Typ-2**-Treiber kommuniziert über eine plattformspezifische Programmbibliothek auf dem Client mit dem Datenbankserver. Das bedeutet, dass für jede Betriebssystem-Plattform zu dem Typ-2-Treiber eine zusätzliche Programmbibliothek benötigt wird.

Mittels des **Typ-3**-Treibers werden die JDBC-API-Befehle in generische DBMS-Befehle übersetzt und (über ein Netzwerkprotokoll) an einen Middleware-Treiber auf einem Anwendungsserver übertragen. Erst dieser Anwendungsserver transformiert die Befehle für die spezifischen Datenbankserver und leitet sie an diese weiter. Ein Typ-3-Treiber benötigt damit keine plattformspezifischen Bibliotheken und muss auch nichts über den verwendeten Datenbankserver wissen.
Typ-3-Treiber eignen sich sehr gut für Internet-Protokolle im Zusammenhang mit Firewalls.

Beim **Typ-4**-Treiber werden die JDBC-API-Befehle direkt in DBMS-Befehle des jeweiligen Datenbankservers übersetzt und (über ein Netzwerkprotokoll) an diesen übertragen. Ein Middleware-Treiber wird dabei nicht verwendet. Damit kann ein Typ-4-Treiber schneller als ein Typ-3-Treiber sein, ist aber weniger flexibel.
Typ-4-Treiber eignen sich gut für Intranet-Lösungen, die schnelle Netzprotokolle nutzen wollen.

3.2 Ablauf

Hier ist der Ablauf eines JDBC-Programmes

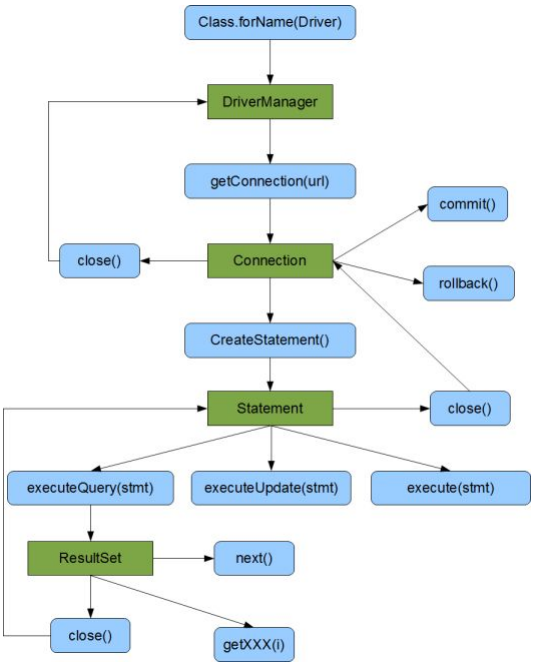

Figure 1: Ablauf einer JDBC-App

3.3 Statements

Ein Statement ist vergleichbar mit einer SQL-Anweisung. Bei JDBC bekommt diese aus einer Connection Instanz.

3.3.1 Statement

Connection.createStatement gibt eine neues Statement zurück und in diesem gibt es dann execute([nichts]/Query/Update) diese nehmen als Parameter SQL in Form von einem String.

3.3.2 PreparedStatement

PreparedStatements sind sehr praktisch wenn man immer wieder ähnliche Abfragen machen muss mit geänderten Parametern in der Where-Clause.
(con.prepareStatement("SELECT t.* FROM t_table t
WHERE t.p1 = ?").setString(1,"Teststring")).executeQuery();

3.3.3 CallableStatement

CallableStatement ermöglicht das Aufrufen von Stored Procedures

4 e-SQL

Embedded SQL ist eine Möglichkeit mit den Daten einer Datenbank zu arbeiten. Für e-SQL Statements wird ein PreCompiler benötigt, denn dieser wandelt das Statement in den benltigten Code der Hochsprache um. Zu dem Programm muss die Bibliothek / Library gelinkt / referenziert werden. Die Bibliothek ist abhängig von der verwendeten Datenbank und wird gewöhnlich vom jeweiligen Datenbankhersteller geliefert.
Prinzipiell sollten Programme mit e-SQL kompatibel mit mehreren Datenbankherstellern sein. Jedoch können auch schon nur Versions updates veraussetzen, dass man das Programm neu kompiliert.

Der Vorteil dieses Ansatzes der Datenbankanbindung liegt darin, dass zur Compilezeit nicht nur die SQL-Syntax sondern auch die Typverträglichkeit der Schnittstellenvariablen gegen die Datentypen des Datenbanksystems geprüft werden kann. Laufzeitgebundene Architekturen wie ODBC und JDBC können keine Typprüfung vornehmen.

4.1 Beispiel

```
#sql { <sql-statement> }; //syntax

#sql{
SELECT vorname, nachname
INTO :vorname, :nachname //Programmvariablen haben einen : davor
FROM mitarbeitertabelle
WHERE pnr = :pnr
};
```

5 DAO

Data-Access-Object ist ein Interface bzw. ein Objekt, das Zugriff auf die Datenbank oder andere Speicherplätze. Es besitzt Parallelen zu ORM's, denn eine Klasse representiert eine Entität.

5.1 Beispiel

Abbildung der Entität Employee [2]

```
public class Employee {
    private int id;
    private String name;

    public int getId() {
        return id;
    }
    public void setId(int id) {
        this.id = id;
    }
    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }
}
```

DAO-Interface für die Klasse Employee [2]

```
interface EmployeeDAO {
    List<Employee> findAll();
    List<Employee> findById(int id);
    List<Employee> findByName(String name);
    boolean insertEmployee(Employee employee);
    boolean updateEmployee(Employee employee);
    boolean deleteEmployee(Employee employee);
}
```

Sinn von DAO ist es Zugriffsmethodik von Daten zu trennen.

6 Quellen

[1] http://edb.gm.fh-koeln.de/jdbc_trainer/javatrainer/de/html/unit_einfuehrung.html
[2] http://stackoverflow.com/questions/19154202/data-access-object-dao-in-java